HTML

1. HTML Introduction 2
2. HTML Basic Examples 4
HTML Paragraphs 5
3. HTML Elements 7
4. HTML Attributes 9
5. HTML Headings 12
6. HTML Paragraphs 13
7. HTML Styles 14
8. HTML Text Formatting 16
9. HTML Quotation and Citation Elements 19
10. HTML Colors 22
11. HTML Links - Hyperlinks 24
12. HTML Link Colors 25
13. HTML Images 29
14. HTML Tables 33
15. HTML Lists 37
16. HTML Block and Inline Elements 42
17. HTML The class Attribute 44
18. HTML Forms 45
19. Access Key, Data, hidden and tab index 49
20. Preparing audio and video 51
21 - Block-level elements, inline elements and content models 52
22 - The Semantic Web 53

CSS 54

Java Script 78

1. HTML Introduction

HTML is the standard markup language for creating Web pages.

- HTML stands for Hyper Text Markup Language
- HTML describes the structure of Web pages using markup
- HTML elements are the building blocks of HTML pages
- HTML elements are represented by tags
- HTML tags label pieces of content such as "heading", "paragraph", "table", and so on
- Browsers do not display the HTML tags, but use them to render the content of the page

Example Explained

- The `<!DOCTYPE html>` declaration defines this document to be HTML5
- The `<html>` element is the root element of an HTML page
- The `<head>` element contains meta information about the document
- The `<title>` element specifies a title for the document
- The `<body>` element contains the visible page content
- The `<h1>` element defines a large heading
- The `<p>` element defines a paragraph

```
<!DOCTYPE html>
<html>
<head>
<title>Page Title</title>
</head>
<body>

<h1>My First Heading</h1>
<p>My first paragraph.</p>

</body>
</html
```

HTML Tags

HTML tags are element names surrounded by angle brackets:

<tagname>content goes here...</tagname>

- HTML tags normally come **in pairs** like `<p>` and `</p>`
- The first tag in a pair is the **start tag,** the second tag is the **end tag**
- The end tag is written like the start tag, but with a **forward slash** inserted before the tag name

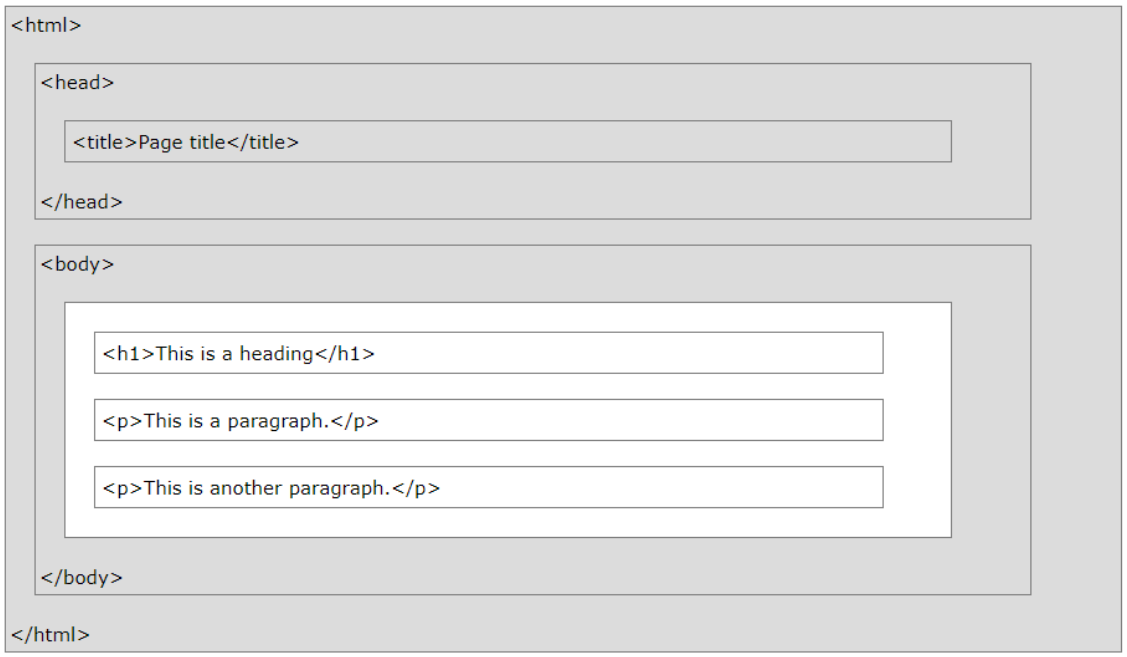
Tip: The start tag is also called the **opening tag**, and the end tag the **closing tag**.

Web Browsers

The purpose of a web browser (Chrome, IE, Firefox, Safari, etc.) is to read HTML documents and display them. The browser does not display the HTML tags, but uses them to determine how to display the document:

HTML Page Structure

Below is a visualization of an HTML page structure:



Note: Only the content inside the <body> section (the white area above) is displayed in a browser.

The <!DOCTYPE> Declaration

The `<!DOCTYPE>` declaration represents the document type, and helps browsers to display web pages correctly.

It must only appear once, at the top of the page (before any HTML tags).

The `<!DOCTYPE>` declaration is not case sensitive.

The `<!DOCTYPE>` declaration for HTML5 is: `<!DOCTYPE html>`

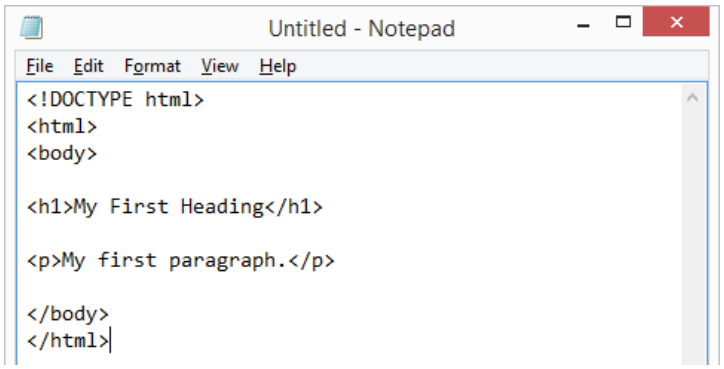
HTML Versions

Since the early days of the web, there have been many versions of HTML:

Version	Year				
HTML	1991	HTML 2.0	1995	HTML 3.2	1997
HTML 4.01	1999	XHTML	2000	**HTML5**	**2014**

Write HTML Using Notepad

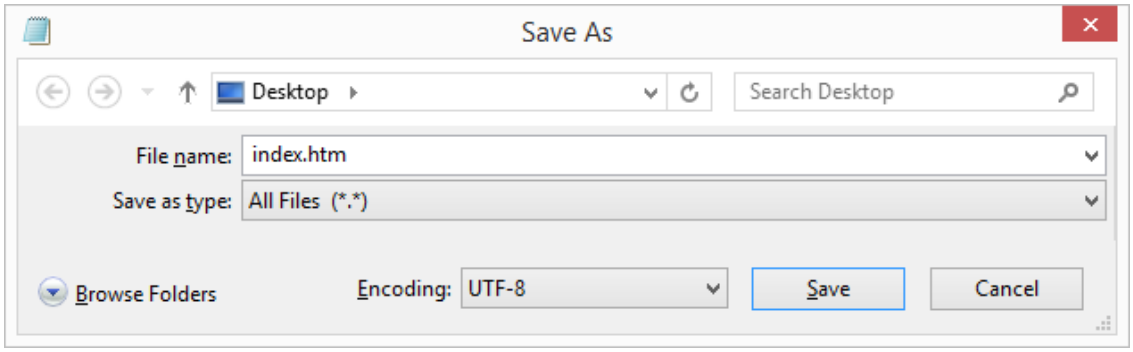
Let's use a simple text editor to learn HTML.



Step 3: Save the HTML Page

Save the file on your computer. Select **File > Save as** in the Notepad menu.

Name the file **"index.htm"** and set the encoding to **UTF-8** (which is the preferred encoding for HTML files).



You can use either .htm or .html as file extension. There is no difference, it is up to you.

2. HTML Basic Examples

HTML Documents

All HTML documents must start with a document type declaration: `<!DOCTYPE html>`.

The HTML document itself begins with `<html>` and ends with `</html>`.

The visible part of the HTML document is between `<body>` and `</body>`.

```
<!DOCTYPE html>
<html>
<body>

<h1>My First Heading</h1>
<p>My first paragraph.</p>

</body>
</html>
```

HTML Headings

HTML headings are defined with the `<h1>` to `<h6>` tags.

`<h1>` defines the most important heading. `<h6>` defines the least important heading:

```
<h1>This is heading 1</h1>
<h2>This is heading 2</h2>
<h3>This is heading 3</h3>
```

HTML Paragraphs

HTML paragraphs are defined with the `<p>` tag:

```
<p>This is a paragraph.</p>
<p>This is another paragraph.</p>
```

HTML Links

HTML links are defined with the `<a>` tag:

```
<a href="https://www.w3schools.com">This is a link</a>
```

- The link's destination is specified in the `href` attribute.
- Attributes are used to provide additional information about HTML elements.

HTML Images

HTML images are defined with the `<img>` tag.

The source file (`src`), alternative text (`alt`), `width`, and `height` are provided as attributes:

```
<img src="w3schools.jpg" alt="W3Schools.com" width="104" height="142">
```

HTML Buttons

HTML buttons are defined with the `<button>` tag:

```
<button>Click me</button>
```

HTML Lists

HTML lists are defined with the `<ul>` (unordered/bullet list) or the `<ol>` (ordered/numbered list) tag, followed by `<li>` tags (list items):

```
<ul>
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ul>
```

Try the code:

```
<!DOCTYPE html>
<html>
<body>

<h2>An Unordered HTML List</h2>

<ul>
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ul>

<h2>An Ordered HTML List</h2>

<ol>
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ol>

</body>
</html>
```

An Unordered HTML List

- Coffee
- Tea
- Milk

An Ordered HTML List

1. Coffee
2. Tea
3. Milk

3. HTML Elements

An HTML element usually consists of a **start** tag and **end** tag, with the content inserted in between:

<tagname>Content goes here...</tagname>

The HTML **element** is everything from the start tag to the end tag:

My first paragraph.

HTML elements with no content are called empty elements. Empty elements do not have an end tag, such as the <br> element (which indicates a line break).

Nested HTML Elements

- HTML elements can be nested (elements can contain elements).
- All HTML documents consist of nested HTML elements.
- This example contains four HTML elements:

```
<!DOCTYPE html>
<html>
<body>

<h1>My First
Heading</h1>
<p>My first
paragraph.</p>

</body></html>
```

Example Explained

- The `<html>` element defines the **whole document**.
  - It has a **start** tag <html> and an **end** tag </html>.
- The element **content** is another HTML element (the `<body>` element).
  - The `<body>` element defines the **document body**.
  - It has a **start** tag <body> and an **end** tag </body>.

- The element **content** is two other HTML elements (`<h1>` and `<p>`).
- The `<h1>` element defines a **heading**.
  - It has a **start** tag <h1> and an **end** tag </h1>.
  - The element **content** is: My First Heading.
- The `<p>` element defines a **paragraph**.
  - It has a **start** tag and an **end** tag .
  - The element **content** is: My first paragraph.

Do Not Forget the End Tag

Some HTML elements will display correctly, even if you forget the end tag:

Empty HTML Elements

HTML elements with no content are called empty elements.

`<br>` is an empty element without a closing tag (the `<br>` tag defines a line break).

Empty elements can be "closed" in the opening tag like this: <br />.

HTML5 does not require empty elements to be closed. But if you want stricter validation, or if you need to make your document readable by XML parsers, you must close all HTML elements properly.

Use Lowercase Tags

HTML tags are not case sensitive: <P> means the same as <p>.

The HTML5 standard does not require lowercase tags, but W3C **recommends** lowercase in HTML, and **demands** lowercase for stricter document types like XHTML.

4. HTML Attributes

Attributes provide additional information about HTML elements.

- All HTML elements can have **attributes**
- Attributes provide **additional information** about an element
- Attributes are always specified in **the start tag**
- Attributes usually come in name/value pairs like: **name="value"**

The href Attribute

HTML links are defined with the `<a>` tag. The link address is specified in the `href` attribute:

```
<a href="https://www.w3schools.com">This is a link</a>
```

The src Attribute

HTML images are defined with the `<img>` tag.

The filename of the image source is specified in the `src` attribute:

```
<img src="img_girl.jpg">
```

The width and height Attributes

Images in HTML have a set of **size** attributes, which specifies the width and height of the image:

```
<img src="img_girl.jpg" width="500" height="600">
```

The image size is specified in pixels: width="500" means 500 pixels wide.

The alt Attribute

The `alt` attribute specifies an alternative text to be used, when an image cannot be displayed.

The value of the attribute can be read by screen readers. This way, someone "listening" to the webpage, e.g. a blind person, can "hear" the element.

```
<img src="img_girl.jpg" alt="Girl with a jacket">
```

The `alt` attribute is also useful if the image does not exist: What happens if we try to display an image that does not exist?

The style Attribute

The `style` attribute is used to specify the styling of an element, like color, font, size etc.

```
<p style="color:red">I am a paragraph</p>
```

The lang Attribute

The language of the document can be declared in the `<html>` tag.

The language is declared with the `lang` attribute.

Declaring a language is important for accessibility applications (screen readers) and search engines:

```
<!DOCTYPE html>
<html lang="en-US">
<body>

...

</body>
</html>
```

The first two letters specify the language (en). If there is a dialect, use two more letters (US).

The title Attribute

Here, a `title` attribute is added to the `<p>` element. The value of the title attribute will be displayed as a tooltip when you mouse over the paragraph:

```
<p title="I'm a tooltip">
This is a paragraph.
</p>
```

We Suggest: Use Lowercase Attributes

We Suggest: Quote Attribute Values

The HTML5 standard does not require quotes around attribute values.

The `href` attribute, demonstrated above, *can* be written without quotes:

Bad: <a href=https://www.w3schools.com>

Good: <a href="https://www.w3schools.com">

W3C **recommends** quotes in HTML, and **demands** quotes for stricter document types like XHTML. Sometimes it is **necessary** to use quotes. This example will not display the title attribute correctly, because it contains a space:

```
<p title=About W3Schools>
```

Using quotes are the most common. Omitting quotes can produce errors.

Single or Double Quotes?

Double quotes around attribute values are the most common in HTML, but single quotes can also be used. In some situations, when the attribute value itself contains double quotes, it is necessary to use single quotes:

```
<p title='John "ShotGun" Nelson'>
```

Or vice versa:

```
<p title="John 'ShotGun' Nelson">
```

Chapter Summary

- All HTML elements can have **attributes**
- The `title` attribute provides additional "tool-tip" information
- The `href` attribute provides address information for links

- The `width` and `height` attributes provide size information for images
- The `alt` attribute provides text for screen readers
- At W3Schools we always use **lowercase** attribute names
- At W3Schools we always **quote** attribute values with double quotes

Below is an alphabetical list of some attributes often used in HTML:

Attribute	Description
alt	Specifies an alternative text for an image, when the image cannot be displayed
disabled	Specifies that an input element should be disabled
href	Specifies the URL (web address) for a link
id	Specifies a unique id for an element
src	Specifies the URL (web address) for an image
style	Specifies an inline CSS style for an element
title	Specifies extra information about an element (displayed as a tool tip)

5. HTML Headings

Headings are defined with the `<h1>` to `<h6>` tags.

`<h1>` defines the most important heading. `<h6>` defines the least important heading.

Why are Headings Important?

Search engines use the headings to index the structure and content of your web pages. Users skim your pages by its headings. It is important to use headings to show the document structure.

`<h1>` headings should be used for main headings, followed by `<h2>` headings, then the less important `<h3>`, and so on.

Note: Use HTML headings for headings only. Don't use headings to make text **BIG** or **bold**.

Bigger Headings

Each HTML heading has a default size. However, you can specify the size for any heading with the `style` attribute, using the CSS `font-size` property:

```
<h1 style="font-size:60px;">Heading 1</h1>
```

HTML Horizontal Rules

The `<hr>` tag defines a thematic break in an HTML page, and is most often displayed as a horizontal rule.

The `<hr>` element is used to separate content (or define a change) in an HTML page:

```
<h1>This is heading 1</h1>
<p>This is some text.</p>
<hr>
<h2>This is heading 2</h2>
<p>This is some other text.</p>
<hr>
```

The HTML <head> Element

The HTML `<head>` element has nothing to do with HTML headings.

The `<head>` element is a container for metadata. HTML metadata is data about the HTML document. Metadata is not displayed.

The `<head>` element is placed between the `<html>` tag and the `<body>` tag:

```
<!DOCTYPE html>
<html>
<head>
  <title>My First HTML</title>
  <meta charset="UTF-8">
</head>
<body>

.

.
```

View HTML Source Code:

Right-click in an HTML page and select "View Page Source" (in Chrome) or "View Source" (in IE), or similar in other browsers. This will open a window containing the HTML source code of the page.

Inspect an HTML Element:

Right-click on an element (or a blank area), and choose "Inspect" or "Inspect Element" to see what elements are made up of (you will see both the HTML and the CSS). You can also edit the HTML or CSS on-the-fly in the Elements or Styles panel that opens.

6. HTML Paragraphs

The HTML `<p>` element defines a **paragraph**:

You cannot be sure how HTML will be displayed. Large or small screens, and resized windows will create different results. With HTML, you cannot change the output by adding extra spaces or extra lines in your HTML code. The browser will remove any extra spaces and extra lines when the page is displayed:

```
<p>
This paragraph
contains          a lot of spaces
in the source          code,
but the        browser
ignores it.
</p>
```

HTML Line Breaks

- The HTML `<br>` element defines a **line break**.
- Use `<br>` if you want a line break (a new line) without starting a new paragraph:
- The `<br>` tag is an empty tag, which means that it has no end tag.

```
<p>This is<br>a paragraph<br>with line breaks.</p>
```

- Write a poem/song lyrics with each sentence in a new line.

The HTML <pre> Element

The HTML `<pre>` element defines preformatted text.

The text inside a `<pre>` element is displayed in a fixed-width font (usually Courier), and it preserves both spaces and line breaks:

```
<!DOCTYPE html> <html> <body>

<p>The pre tag preserves both spaces and line breaks:</p>

<pre>

My Bonnie lies over the ocean.


My Bonnie lies over the sea.


My Bonnie lies over the ocean.


Oh, bring back my Bonnie to me.

</pre> </body>  </html>
```

7. HTML Styles

I am Red

I am Blue

I am Big

The HTML Style Attribute

Setting the style of an HTML element, can be done with the `style` attribute.

The HTML `style` attribute has the following **syntax**:

```
<tagname style="property:value;">
```

The ***property*** is a CSS property. The ***value*** is a CSS value.

HTML Background Color

The `background-color` property defines the background color for an HTML element.

This example sets the background color for a page to powderblue:

```
<body style="background-color:powderblue;">

<h1>This is a heading</h1>
<p>This is a paragraph.</p>

</body>
```

HTML Text Color

The `color` property defines the text color for an HTML element:

```
<h1 style="color:blue;">This is a heading</h1>
<p style="color:red;">This is a paragraph.</p>
```

HTML Fonts

The `font-family` property defines the font to be used for an HTML element:

```
<h1 style="font-family:verdana;">This is a heading</h1>
<p style="font-family:courier;">This is a paragraph.</p>
```

HTML Text Size

The `font-size` property defines the text size for an HTML element:

```
<h1 style="font-size:300%;">This is a heading</h1>
<p style="font-size:160%;">This is a paragraph.</p>
```

HTML Text Alignment

The `text-align` property defines the horizontal text alignment for an HTML element:

```
<h1 style="text-align:center;">Centered Heading</h1>
<p style="text-align:center;">Centered paragraph.</p>
```

Chapter Summary

- Use the `style` attribute for styling HTML elements
- Use `background-color` for background color
- Use `color` for text colors
- Use `font-family` for text fonts
- Use `font-size` for text sizes
- Use `text-align` for text alignment

8. HTML Text Formatting

Text Formatting

This text is bold

This text is italic

This is subscript and superscript

In the previous chapter, you learned about the HTML **style attribute**.

HTML also defines special **elements** for defining text with a special **meaning**.

HTML uses elements like `<b>` and `<i>` for formatting output, like **bold** or *italic* text.

Formatting elements were designed to display special types of text:

- `<b>` - Bold text
- `<strong>` - Important text
- `<i>` - Italic text
- `<em>` - Emphasized text
- `<mark>` - Marked text
- `<small>` - Small text
- `<del>` - Deleted text
- `<ins>` - Inserted text
- `<sub>` - Subscript text
- `<sup>` - Superscript text

HTML <b> and Elements

The HTML `<b>` element defines **bold** text, without any extra importance.

```
<b>This text is bold</b>
```

The HTML `<strong>` element defines **strong** text, with added semantic "strong" importance.

```
<strong>This text is strong</strong>
```

HTML <i> and Elements

The HTML `<i>` element defines *italic* text, without any extra importance.

```
<i>This text is italic</i>
```

The HTML `<em>` element defines *emphasized* text, with added semantic importance.

```
<em>This text is emphasized</em>
```

Note: Browsers display `<strong>` as `<b>`, and `<em>` as `<i>`. However, there is a difference in the meaning of these tags: `<b>` and `<i>` defines bold and italic text, but `<strong>` and `<em>` means that the text is "important".

HTML <small> Element

The HTML `<small>` element defines smaller text:

```
<h2>HTML <small>Small</small> Formatting</h2>
```

HTML <mark> Element

The HTML `<mark>` element defines marked or highlighted text:

```
<h2>HTML <mark>Marked</mark> Formatting</h2>
```

HTML <del> Element

The HTML `<del>` element defines ~~deleted~~ (removed) text.

```
<p>My favorite color is <del>blue</del> red.</p>
```

HTML <ins> Element

The HTML `<ins>` element defines inserted (added) text.

```
<p>My favorite <ins>color</ins> is red.</p>
```

HTML <sub> Element

The HTML `<sub>` element defines $_{\text{subscripted}}$ text.

```
<p>This is <sub>subscripted</sub> text.</p>
```

HTML <sup> Element

The HTML `<sup>` element defines $^{\text{superscripted}}$ text.

```
<p>This is <sup>superscripted</sup> text.</p>
```

Tag	Description
<b>	Defines bold text
	Defines emphasized text
<i>	Defines italic text
<small>	Defines smaller text
	Defines important text

<sub>	Defines subscripted text
<sup>	Defines superscripted text
<ins>	Defines inserted text
<del>	Defines deleted text
<mark>	Defines marked/highlighted text

9. HTML Quotation and Citation Elements

HTML <q> for Short Quotations

The HTML `<q>` element defines a short quotation.

Browsers usually insert quotation marks around the `<q>` element.

```
<p>WWF's goal is to: <q>Build a future where people live in harmony
with nature.</q></p>
```

HTML <blockquote> for Quotations

The HTML `<blockquote>` element defines a section that is quoted from another source.

Browsers usually indent `<blockquote>` elements.

```
<p>Here is a quote from WWF's website:</p>
<blockquote cite="http://www.worldwildlife.org/who/index.html">
For 50 years, WWF has been protecting the future of nature.
The world's leading conservation organization,
WWF works in 100 countries and is supported by
1.2 million members in the United States and
close to 5 million globally.
</blockquote>
```

HTML <abbr> for Abbreviations

The HTML `<abbr>` element defines an abbreviation or an acronym.

Marking abbreviations can give useful information to browsers, translation systems and search-engines.

```
<p>The <abbr title="World Health Organization">WHO</abbr> was founded in 1948.</p>
```

HTML <address> for Contact Information

The HTML `<address>` element defines contact information (author/owner) of a document or an article.

The `<address>` element is usually displayed in italic. Most browsers will add a line break before and after the element.

```
<address>
Written by John Doe.<br>
Visit us at:<br>
Example.com<br>
Box 564, Disneyland<br>
USA
</address>
```

HTML <cite> for Work Title

The HTML `<cite>` element defines the title of a work.

Browsers usually display `<cite>` elements in italic.

```
<p><cite>The Scream</cite> by Edvard Munch. Painted in 1893.</p>
```

HTML <bdo> for Bi-Directional Override

The HTML `<bdo>` element defines bi-directional override.

The `<bdo>` element is used to override the current text direction:

```
<bdo dir="rtl">This text will be written from right to left</bdo>
```

Tag	Description
<abbr>	Defines an abbreviation or acronym
<address>	Defines contact information for the author/owner of a document
<bdo>	Defines the text direction
<blockquote>	Defines a section that is quoted from another source
<cite>	Defines the title of a work
<q>	Defines a short inline quotation

HTML Comment Tags

You can add comments to your HTML source by using the following syntax:

```
<!-- Write your comments here -->
```

Comments are also great for debugging HTML, because you can comment out HTML lines of code, one at a time, to search for errors:

10. HTML Colors

HTML colors are specified using predefined color names, or RGB, HEX, HSL, RGBA, HSLA values.

Background Color

You can set the background color for HTML elements:

```
<h1 style="background-color:DodgerBlue;">Hello World</h1>
<p style="background-color:Tomato;">Lorem ipsum...</p>
```

Text Color

You can set the color of text:

```
<h1 style="color:Tomato;">Hello World</h1>
<p style="color:DodgerBlue;">Lorem ipsum...</p>
<p style="color:MediumSeaGreen;">Ut wisi enim...</p>
```

Border Color

You can set the color of borders

```
<h1 style="border:2px solid Tomato;">Hello World</h1>
<h1 style="border:2px solid DodgerBlue;">Hello World</h1>
<h1 style="border:2px solid Violet;">Hello World</h1>
```

Color Values

In HTML, colors can also be specified using RGB values, HEX values, HSL values, RGBA values, and HSLA values:

```
<h1 style="background-color:rgb(255, 99, 71);">...</h1>
<h1 style="background-color:#ff6347;">...</h1>
<h1 style="background-color:hsl(9, 100%, 64%);">...</h1>

<h1 style="background-color:rgba(255, 99, 71, 0.5);">...</h1>
<h1 style="background-color:hsla(9, 100%, 64%, 0.5);">...</h1>
```

RGB Value

In HTML, a color can be specified as an RGB value, using this formula:

rgb(*red, green, blue*)

Each parameter (red, green, and blue) defines the intensity of the color between 0 and 255.

For example, rgb(255, 0, 0) is displayed as red, because red is set to its highest value (255) and the others are set to 0.

To display the color black, all color parameters must be set to 0, like this: rgb(0, 0, 0).

To display the color white, all color parameters must be set to 255, like this: rgb(255, 255, 255).

HEX Value

In HTML, a color can be specified using a hexadecimal value in the form:

#rrggbb

Where rr (red), gg (green) and bb (blue) are hexadecimal values between 00 and ff (same as decimal 0-255).

For example, #ff0000 is displayed as red, because red is set to its highest value (ff) and the others are set to the lowest value (00).

HSL Value

In HTML, a color can be specified using hue, saturation, and lightness (HSL) in the form:

hsl(*hue, saturation, lightness*)

Hue is a degree on the color wheel from 0 to 360. 0 is red, 120 is green, and 240 is blue.

Saturation is a percentage value, 0% means a shade of gray, and 100% is the full color.

Lightness is also a percentage, 0% is black, 50% is neither light or dark, 100% is white

Saturation

- Saturation can be described as the intensity of a color.
- 100% is pure color, no shades of gray
- 50% is 50% gray, but you can still see the color.
- 0% is completely gray, you can no longer see the color.

Lightness

The lightness of a color can be described as how much light you want to give the color, where 0% means no light (black), 50% means 50% light (neither dark nor light) 100% means full lightness (white).

Shades of gray are often defined by setting the hue and saturation to 0, and adjust the lightness from 0% to 100% to get darker/lighter shades:

RGBA Value

RGBA color values are an extension of RGB color values with an alpha channel - which specifies the opacity for a color. An RGBA color value is specified with:

rgba(*red, green, blue, alpha*)

The alpha parameter is a number between 0.0 (fully transparent) and 1.0 (not transparent at all):

11. HTML Links - Hyperlinks

HTML links are hyperlinks.

You can click on a link and jump to another document.

When you move the mouse over a link, the mouse arrow will turn into a little hand.

Note: A link does not have to be text. It can be an image or any other HTML element.

HTML Links - Syntax

In HTML, links are defined with the `<a>` tag:

```
<a href="url">link text</a>
```

```
<a href="https://www.w3schools.com/html/">Visit our HTML tutorial</a>
```

The **link text** is the visible part (Visit our HTML tutorial).

Clicking on the link text will send you to the specified address.

Note: Without a forward slash at the end of subfolder addresses, you might generate two requests to the server. Many servers will automatically add a forward slash to the end of the address, and then create a new request.

Local Links

The example above used an absolute URL (a full web address).

A local link (link to the same web site) is specified with a relative URL (without http://www....).

```
<a href="html_images.asp">HTML Images</a>
```

12. HTML Link Colors

By default, a link will appear like this (in all browsers):

- An unvisited link is underlined and blue
- A visited link is underlined and purple
- An active link is underlined and red

You can change the default colors, by using CSS:

```
<style>
a:link {
    color: green;
    background-color: transparent;
    text-decoration: none;
}

a:visited {
    color: pink;
    background-color: transparent;
    text-decoration: none;
}

a:hover {
    color: red;
    background-color: transparent;
    text-decoration: underline;
}

a:active {
    color: yellow;
    background-color: transparent;
    text-decoration: underline;
}
</style>
```

HTML Links - The target Attribute

The `target` attribute specifies where to open the linked document.

The target attribute can have one of the following values:

- _blank - Opens the linked document in a new window or tab
- _self - Opens the linked document in the same window/tab as it was clicked (this is default)
- _parent - Opens the linked document in the parent frame
- _top - Opens the linked document in the full body of the window
- *framename* - Opens the linked document in a named frame

This example will open the linked document in a new browser window/tab:

```
<a href="https://www.w3schools.com/html/" target="_top">HTML5
tutorial!</a>
```

HTML Links - Image as Link

It is common to use images as links:

```
<a href="default.asp">
  <img src="smiley.gif" alt="HTML
tutorial" style="width:42px;height:42px;border:0;">
</a>
```

Note: `border:0;` is added to prevent IE9 (and earlier) from displaying a border around the image (when the image is a link).

Link Titles

The `title` attribute specifies extra information about an element. The information is most often shown as a tooltip text when the mouse moves over the element.

```
<a href="https://www.w3schools.com/html/" title="Go to W3Schools HTML
section">Visit our HTML Tutorial</a>
```

HTML Links - Create a Bookmark

HTML bookmarks are used to allow readers to jump to specific parts of a Web page. Bookmarks can be useful if your webpage is very long. To make a bookmark, you must first create the bookmark, and then add a link to it. When the link is clicked, the page will scroll to the location with the bookmark.

Example

First, create a bookmark with the `id` attribute:

```
<h2 id="C4">Chapter 4</h2>
```

Then, add a link to the bookmark ("Jump to Chapter 4"), from within the same page:

```
<a href="#C4">Jump to Chapter 4</a>
```

Or, add a link to the bookmark ("Jump to Chapter 4"), from another page:

```
<a href="html_demo.html#C4">Jump to Chapter 4</a>
```

External Paths

External pages can be referenced with a full URL or with a path relative to the current web page.

This example uses a full URL to link to a web page:

```
<a href="https://www.w3schools.com/html/default.asp">HTML tutorial</a>
```

This example links to a page located in the html folder on the current web site:

```
<a href="/html/default.asp">HTML tutorial</a>
```

This example links to a page located in the same folder as the current page:

```
<a href="default.asp">HTML tutorial</a>
```

Chapter Summary

- Use the `<a>` element to define a link
- Use the `href` attribute to define the link address
- Use the `target` attribute to define where to open the linked document
- Use the `<img>` element (inside <a>) to use an image as a link
- Use the `id` attribute (id="*value*") to define bookmarks in a page
- Use the `href` attribute (href="#*value*") to link to the bookmark

13. HTML Images

Images can improve the design and the appearance of a web page

```
<img src="pic_trulli.jpg" alt="Italian Trulli">
```

HTML Images Syntax

- In HTML, images are defined with the `<img>` tag.
- The `<img>` tag is empty, it contains attributes only, and does not have a closing tag.
- The `src` attribute specifies the URL (web address) of the image:
- `<img src="url">`

The alt Attribute

The `alt` attribute provides an alternate text for an image, if the user for some reason cannot view it (because of slow connection, an error in the src attribute, or if the user uses a screen reader).

The value of the `alt` attribute should describe the image:

mage Size - Width and Height

You can use the `style` attribute to specify the width and height of an image.

```
<img src="img_girl.jpg" alt="Girl in a
jacket" style="width:500px;height:600px;">
```

Alternatively, you can use the `width` and `height` attributes:

```
<img src="img_girl.jpg" alt="Girl in a jacket" width="500" height="600">
```

The `width` and `height` attributes always defines the width and height of the image in pixels.

Width and Height, or Style?

The `width`, `height`, and `style` attributes are valid in HTML5.

However, we suggest using the `style` attribute. It prevents styles sheets from changing the size of images:

Images in Another Folder

If not specified, the browser expects to find the image in the same folder as the web page.

However, it is common to store images in a sub-folder. You must then include the folder name in the `src` attribute:

```
<img src="/images/html5.gif" alt="HTML5
Icon" style="width:128px;height:128px;">
```

Images on Another Server

Some web sites store their images on image servers.

Actually, you can access images from any web address in the world:

```
<img src="https://www.w3schools.com/images/w3schools_green.jpg" alt="W3Sch
ools.com">
```

Image as a Link

To use an image as a link, put the `<img>` tag inside the `<a>` tag

```
<a href="default.asp">
  <img src="smiley.gif" alt="HTML
tutorial" style="width:42px;height:42px;border:0;">
</a>
```

Image Maps

The `<map>` tag defines an image-map. An image-map is an image with clickable areas.

In the image below, click on the computer, the phone, or the cup of coffee:

```
<img src="workplace.jpg" alt="Workplace" usemap="#workmap">

<map name="workmap">
  <area shape="rect" coords="34,44,270,350" alt="Computer" href="computer.htm">
  <area shape="rect" coords="290,172,333,250" alt="Phone" href="phone.htm">
  <area shape="circle" coords="337,300,44" alt="Coffee" href="coffee.htm">
</map>
```

The `name` attribute of the `<map>` tag is associated with the `<img>`'s usemap attribute and creates a relationship between the image and the map.

The `<map>` element contains a number of `<area>` tags, that define the clickable areas in the image-map.

Background Image

To add a background image on an HTML element, use the CSS property `background-image`:

To add a background image on a web page, specify the background-image property on the BODY element:

```
<body style="background-image:url('clouds.jpg')">

<h2>Background Image</h2></body>
```

To add a background image on a paragraph, specify the background-image property on the P element:

```
<body>

<p style="background-image:url('clouds.jpg')">
...
</p>

</body>
```

The <picture> Element

HTML5 introduced the `<picture>` element to add more flexibility when specifying image resources.

The `<picture>` element contains a number of <source> elements, each referring to different image sources. This way the browser can choose the image that best fits the current view and/or device.

Each `<source>` element have attributes describing when their image is the most suitable.

The browser will use the first `<source>` element with matching attribute values, and ignore any following `<source>`elements.

Show one picture if the browser window (viewport) is a minimum of 650 pixels, and another image if not, but larger than 465 pixels.

```
<picture>
  <source media="(min-width: 650px)" srcset="img_pink_flowers.jpg">
  <source media="(min-width: 465px)" srcset="img_white_flower.jpg">
  <img src="img_orange_flowers.jpg" alt="Flowers" style="width:auto;
">
</picture>
```

14. HTML Tables

Defining an HTML Table

An HTML table is defined with the `<table>` tag.

Each table row is defined with the `<tr>` tag. A table header is defined with the `<th>` tag. By default, table headings are bold and centered. A table data/cell is defined with the `<td>` tag.

Example

```
<table style="width:100%">
  <tr>
    <th>Firstname</th>
    <th>Lastname</th>
    <th>Age</th>
  </tr>
  <tr>
    <td>Jill</td>
    <td>Smith</td>
    <td>50</td>
  </tr>
  <tr>
    <td>Eve</td>
    <td>Jackson</td>
    <td>94</td>
  </tr>
</table>
```

HTML Table - Adding a Border

If you do not specify a border for the table, it will be displayed without borders.

A border is set using the CSS `border` property:

```
table, th, td {
    border: 1px solid black;
}
```

HTML Table - Collapsed Borders

If you want the borders to collapse into one border, add the CSS `border-collapse` property:

```
table, th, td {
    border: 1px solid black;
    border-collapse: collapse;
}
```

HTML Table - Adding Cell Padding

Cell padding specifies the space between the cell content and its borders. If you do not specify a padding, the table cells will be displayed without padding.

To set the padding, use the CSS `padding` property:

```
th, td {
    padding: 15px;
}
```

HTML Table - Left-align Headings

By default, table headings are bold and centered.

To left-align the table headings, use the CSS `text-align` property:

```
th {
    text-align: left;
}
```

HTML Table - Adding Border Spacing

Border spacing specifies the space between the cells.

To set the border spacing for a table, use the CSS `border-spacing` property:

```
table {
    border-spacing: 5px;
}
```

HTML Table - Cells that Span Many Columns

To make a cell span more than one column, use the `colspan` attribute:

```
<table style="width:100%">
  <tr>
    <th>Name</th>
    <th colspan="2">Telephone</th>
  </tr>
  <tr>
    <td>Bill Gates</td>
    <td>55577854</td>
    <td>55577855</td>
  </tr>
</table>
```

```
<table style="width:100%">
  <tr>
    <th>Name:</th>
    <td>Bill Gates</td>
  </tr>
  <tr>
    <th rowspan="2">Telephone:</th>
    <td>55577854</td>
  </tr>
  <tr>
    <td>55577855</td>
  </tr>
</table>
```

HTML Table - Cells that Span Many Rows

To make a cell span more than one row, use the `rowspan` attribute:

HTML Table - Adding a Caption

To add a caption to a table, use the `<caption>` tag:

```
<table style="width:100%">
  <caption>Monthly savings</caption>
```

A Special Style for One Table

To define a special style for a special table, add an `id` attribute to the table

Now you can define a special style for this table:

```
table#t01 {
    width: 100%;
    background-color: #f1f1c1;
}
```

```
<table id="t01">
  <tr>
    <th>Firstname</th>
    <th>Lastname</th>
    <th>Age</th>
  </tr>
  <tr>
    <td>Eve</td>
    <td>Jackson</td>
    <td>94</td>
  </tr>
</table>
```

And add more styles:->

```
table#t01 tr:nth-child(even) {
    background-color: #eee;
}
table#t01 tr:nth-child(odd) {
    background-color: #fff;
}
table#t01 th {
    color: white;
    background-color: black;
}
```

Chapter Summary

- Use the HTML `<table>` element to define a table
- Use the HTML `<tr>` element to define a table row
- Use the HTML `<td>` element to define a table data
- Use the HTML `<th>` element to define a table heading
- Use the HTML `<caption>` element to define a table caption
- Use the CSS `border` property to define a border
- Use the CSS `border-collapse` property to collapse cell borders
- Use the CSS `padding` property to add padding to cells
- Use the CSS `text-align` property to align cell text
- Use the CSS `border-spacing` property to set the spacing between cells
- Use the `colspan` attribute to make a cell span many columns
- Use the `rowspan` attribute to make a cell span many rows
- Use the `id` attribute to uniquely define one table

15. HTML Lists

Unordered HTML List

An unordered list starts with the `<ul>` tag. Each list item starts with the `<li>` tag.

The list items will be marked with bullets (small black circles) by default:

```
<ul>
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ul>
```

Unordered HTML List - Choose List Item Marker

The CSS `list-style-type` property is used to define the style of the list item marker:

Value	Description
disc	Sets the list item marker to a bullet (default)
circle	Sets the list item marker to a circle
square	Sets the list item marker to a square
none	The list items will not be marked

Example - Disc

```
<ul style="list-style-type:disc">
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ul>
```

Example - Circle

```
<ul style="list-style-type:circle">
  <li>Coffee</li>
  <li>Tea</li>
```

```
  <li>Milk</li>
</ul>
```

Example - Square

```
<ul style="list-style-type:square">
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ul>
```

Example - None

```
<ul style="list-style-type:none">
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ul>
```

Ordered HTML List

An ordered list starts with the `<ol>` tag. Each list item starts with the `<li>` tag.

The list items will be marked with numbers by default:

```
<ol>
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ol>
```

Ordered HTML List - The Type Attribute

The `type` attribute of the `<ol>` tag, defines the type of the list item marker:

Type	Description
type="1"	The list items will be numbered with numbers (default)
type="A"	The list items will be numbered with uppercase letters
type="a"	The list items will be numbered with lowercase letters
type="I"	The list items will be numbered with uppercase roman numbers
type="i"	The list items will be numbered with lowercase roman numbers

Numbers:

```
<ol type="1">
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ol>
```

Uppercase Letters:

```
<ol type="A">
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ol>
```

Lowercase Letters:

```
<ol type="a">
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ol>
```

Uppercase Roman Numbers:

```
<ol type="I">
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ol>
```

Lowercase Roman Numbers:

```
<ol type="i">
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ol>
```

HTML Description Lists

HTML also supports description lists. A description list is a list of terms, with a description of each term. The `<dl>` tag defines the description list, the `<dt>` tag defines the term (name), and the `<dd>` tag describes each term:

```
<dl>
  <dt>Coffee</dt>
  <dd>- black hot drink</dd>
  <dt>Milk</dt>
  <dd>- white cold drink</dd>
</dl>
```

Nested HTML Lists

List can be nested (lists inside lists):

```
<ul>
  <li>Coffee</li>
  <li>Tea
    <ul>
      <li>Black tea</li>
      <li>Green tea</li>
    </ul>
  </li>
  <li>Milk</li>
</ul>
```

Control List Counting

By default, an ordered list will start counting from 1. If you want to start counting from a specified number, you can use the `start` attribute:

```
<ol start="50">
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ol>
```

Horizontal List with CSS

HTML lists can be styled in many different ways with CSS. One popular way is to style a list horizontally, to create a navigation menu:

```
<!DOCTYPE html>
<html>
<head>
<style>
ul {
    list-style-type: none;
    margin: 0;
    padding: 0;
    overflow: hidden;
    background-color: #333333;
}

li {
    float: left;
}

li a {
    display: block;
    color: white;
    text-align: center;
    padding: 16px;
    text-decoration: none;
}

li a:hover {
    background-color: #111111;
}
</style>
</head>
<body>

<ul>
  <li><a href="#home">Home</a></li>
  <li><a href="#news">News</a></li>
  <li><a href="#contact">Contact</a></li>
  <li><a href="#about">About</a></li>
</ul>
```

```
</body>
</html>
```

Chapter Summary

- Use the HTML `<ul>` element to define an unordered list
- Use the CSS `list-style-type` property to define the list item marker
- Use the HTML `<ol>` element to define an ordered list
- Use the HTML `type` attribute to define the numbering type
- Use the HTML `<li>` element to define a list item
- Use the HTML `<dl>` element to define a description list
- Use the HTML `<dt>` element to define the description term
- Use the HTML `<dd>` element to describe the term in a description list
- Lists can be nested inside lists
- List items can contain other HTML elements
- Use the CSS property `float:left` or `display:inline` to display a list horizontally

16. HTML Block and Inline Elements

Every HTML element has a default display value depending on what type of element it is. The default display value for most elements is block or inline.

Block-level Elements

A block-level element always starts on a new line and takes up the full width available (stretches out to the left and right as far as it can).

The element is a block-level element.

```
<div>Hello</div>
<div>World</div>
```

Inline Elements

An inline element does not start on a new line and only takes up as much width as necessary.

This is an inline element inside a paragraph.

```
<span>Hello</span>
<span>World</span>
```

The Element

The `<div>` element is often used as a container for other HTML elements.

The `<div>` element has no required attributes, but `style`, `class` and `id` are common.

When used together with CSS, the `<div>` element can be used to style blocks of content:

```
<div style="background-color:black;color:white;padding:20px;">
  <h2>London</h2>
  <p>London is the capital city of England. It is the most populous city in the United Kingdom, with a metropolitan area of over 13 million inhabitants.</p>
</div>
```

The Element

The `<span>` element is often used as a container for some text.

The `<span>` element has no required attributes, but `style`, `class` and `id` are common.

When used together with CSS, the `<span>` element can be used to style parts of the text:

```
<h1>My <span style="color:red">Important</span> Heading</h1>
```

HTML Grouping Tags

Tag	Description
	Defines a section in a document (block-level)
	Defines a section in a document (inline)

17. HTML The class Attribute

Using The class Attribute

The `class` attribute specifies one or more class names for an HTML element.

The class name can be used by CSS and JavaScript to perform certain tasks for elements with the specified class name.

In CSS, to select elements with a specific class, write a period (.) character, followed by the name of the class:

Use CSS to style all elements with the class name "city":

```
<style>
.city {
    background-color: tomato;
    color: white;
    padding: 10px;
}
</style>

<h2 class="city">London</h2>
<p>London is the capital of England.</p>

<h2 class="city">Paris</h2>
<p>Paris is the capital of France.</p>

<h2 class="city">Tokyo</h2>
<p>Tokyo is the capital of Japan.</p>
```

18. HTML Forms

The <form> Element

Form controls live inside a <form> element. This element should always carry the **action** attribute and will usually have a **method** and **id** attribute too.

Action - Every <form> element requires an action attribute. Its value is the URL for the page on the server that will receive the information in the form when it is submitted.

Method

Forms can be sent using one of two methods: **get** or **post.**

With the get method, the values from the form are added to the end of the URL specified in the action attribute. The get method is ideal for:

- short forms (such as search boxes)
- when you are just retrieving data from the web server (not sending information that should be added to or deleted from a database)

With the post method the values are sent in what are known as HTTP headers. As a rule of thumb you should use the post method if your form:

- allows users to upload a file
- is very long
- contains sensitive data (e.g. passwords)
- adds information to, or deletes information from, a database

If the method attribute is not used, the form data will be sent using the get method.

```
<form action="/action_page.php" method="get">
```

id

The Id is used to identify the form distinctly from other elements on the page (and

is often used by scripts — such as those that check you have entered information into fields

that require values).

The <input> Element

The `<input>` element is the most important form element. The `<input>` element can be displayed in several ways, depending on the **type** attribute.

<input type="text"> Defines a one-line text input field

<input type="radio"> Defines a radio button (for selecting one of many choices)

<input type="submit"> Defines a submit button (for submitting the form)

<input type="password"> the input text will blocked out.

Text Input

`<input type="text">` defines a one-line input field for **text input**:

```
<form>
  First name:<br>
  <input type="text" name="firstname"><br>
  Last name:<br>
  <input type="text" name="lastname">
</form>
```

If the `type` attribute is omitted, the input field gets the default type: "text".

Radio Button Input

`<input type="radio">` defines a **radio button**.

Radio buttons let a user select ONE of a limited number of choices:

```
<form>
  <input type="radio" name="gender" value="male" checked> Male<br>
  <input type="radio" name="gender" value="female"> Female<br>
  <input type="radio" name="gender" value="other"> Other
</form>
```

The Submit Button

`<input type="submit">` defines a button for **submitting** the form data to a **form-handler**.

The form-handler is typically a server page with a script for processing input data. The form-handler is specified in the form's **action** attribute:

```
<form action="/action_page.php">
  First name:<br>
  <input type="text" name="firstname" value="Mickey"><br>
  Last name:<br>
  <input type="text" name="lastname" value="Mouse"><br><br>
  <input type="submit" value="Submit">
</form>
```

The <select> Element

The `<select>` element defines a **drop-down list**:

The `<option>` elements defines an option that can be selected.

By default, the first item in the drop-down list is selected.

To define a pre-selected option, add the `selected` attribute to the option:

```
<select name="cars">
  <option value="volvo">Volvo</option>
  <option value="saab">Saab</option>
  <option value="fiat">Fiat</option>
  <option value="audi">Audi</option>
</select>
```

```
<option value="fiat" selected>Fiat</option>
```

Visible Values:

Use the `size` attribute to specify the number of visible values:

```
<select name="cars" size="3">
```

Allow Multiple Selections:

Use the `multiple` attribute to allow the user to select more than one value:

```
<select name="cars" size="4" multiple>
```

```
="message" rows="10" cols="30">
The cat was playing in the garden.
</textarea>
```

The <textarea> Element

The `<textarea>` element defines a multi-line input field (**a text area**):

```
<textarea name="message" rows="10" cols="30">
The cat was playing in the garden.
</textarea>
```

The `rows` attribute specifies the visible number of lines in a text area.

The `cols` attribute specifies the visible width of a text area.

The Action Attribute

The `action` attribute defines the action to be performed when the form is submitted. Normally, the form data is sent to a web page on the server when the user clicks on the submit button.

In the example above, the form data is sent to a page on the server called "/action_page.php". This page contains a server-side script that handles the form data: `<form action="/action_page.php">`

If the `action` attribute is omitted, the action is set to the current page.

The Target Attribute

The `target` attribute specifies if the submitted result will open in a new browser tab, a frame, or in the current window.

The default value is "`_self`" which means the form will be submitted in the current window.

To make the form result open in a new browser tab, use the value "`_blank`"

```
<form action="/action_page.php" target="_blank">
```

The Name Attribute

- Each input field must have a name attribute to be submitted.
- If the name attribute is omitted, the data of that input field **will not be sent** at all.

Grouping Form Data with <fieldset>

The `<fieldset>` element is used to group related data in a form.

The `<legend>` element defines a caption for the `<fieldset>` element.

```
<form action="/action_page.php">
  <fieldset>
    <legend>Personal information:</legend>
    First name:<br>
    <input type="text" name="firstname" value="Mickey"><br>
    Last name:<br>
    <input type="text" name="lastname" value="Mouse"><br><br>
    <input type="submit" value="Submit">
  </fieldset>
</form>
```

19. Access Key, Data, hidden and tab index

The accesskey attribute specifies a shortcut key to activate/focus an element.

```
<html> <body>

<a href="https://www.w3schools.com/html" accesskey="h">HTML tutorial</a><br>

<a href="https://www.w3schools.com/css" accesskey="c">CSS tutorial</a>

<p>The accesskey attribute specifies a shortcut key to activate/focus an element.</p>

<p><strong>Note:</strong> The shortcut is varying in different browsers:</p>

<ul>

  <li>IE, Chrome, Safari, Opera 15+: [ALT] + <em>accesskey</em></li>

  <li>Opera prior version 15: [SHIFT] [ESC] + <em>accesskey</em></li>

  <li>Firefox: [ALT] [SHIFT] + <em>accesskey</em></li>

</ul>

</body> </html>
```

The tabindex attribute specifies the tab order of an element (when the "tab" button is used for navigating).

```
<!DOCTYPE html>

<html> <body>

<a href="https://www.w3schools.com/" tabindex="2">W3Schools</a><br>
```

```
<a href="http://www.google.com/" tabindex="1">Google</a><br>

<a href="http://www.microsoft.com/" tabindex="3">Microsoft</a>

<p><b>Note:</b> Try navigating the links by using the "Tab" button on your keyboard.</p>

</body> </html>
```

The hidden attribute is a boolean attribute. When present, it specifies that an element is not yet, or is no longer, relevant. Browsers should not display elements that have the hidden attribute specified.

```
<!DOCTYPE html>

<html><body>

<p hidden>This paragraph should be hidden.</p>

<p>This is a visible paragraph.</p>

<p><b>Note:</b> The hidden attribute is not supported in IE10 or earlier versions.</p>

</body></html>
```

Email Link

```
<p>

This is an email link:

<a href="mailto:razmi@ais.lk?Subject=Hello%20again" target="_top">Send Mail</a>

</p>
```

The <data> tag links the given content with a machine-readable translation.

This element provides both a machine-readable value for data processors, and a human-readable value for rendering in a browser.

```
<!DOCTYPE html>
<html><body>
<p>The following example displays product names but also associates each name with a product number:</p>
<ul>
 <li><data value="21053">Cherry Tomato</data></li>
<li><data value="21054">Beef Tomato</data></li>
 <li><data value="21055">Snack Tomato</data></li>
</ul>
</body></html>
```

20. Preparing audio and video

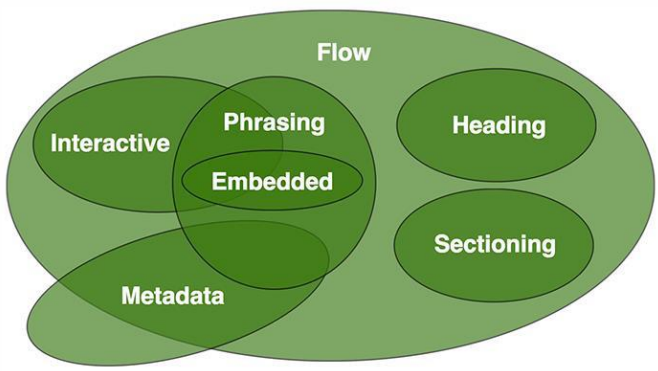
o multiple file formats (wav, mpeg, mp3, mp4, webm, ogg, possible browser playback issues)

- <audio></audio>
- <video></video>
- controls
- <source> (can include multiple sound files in different formats to combat possible browser playback issues)
- type
- autoplay
- loop

```
<audio controls autoplay loop>
<source src="barkingDogs.mp3" type="audio/mpeg">
<source src="barkingDogs.wav type="audio/wav">
</audio>
<video controls width="320" height="240">
<source src="barkingDogs/mp4">
<source src="barkingDogs/ogg ">
</video>
```

21 - Block-level elements, inline elements and content models

- prior to modern HTML specification, HTML elements were either block-level or inline elements
- block level elements
    - take up the entire width of their parent
    - always begin on a new line in the flow of the document
- inline elements
    - are on the same line (could have many inline elements, one after another, and they will all still be displayed on the same line
    - restricted to only contain other inline elements
    - take up only as much space as necessary
- modern content models
    - a content model refers to the set of rules that define what type of content each element can have
    - they can still loosely be thought of as falling into either block-level or inline elements
    - flow content
        - roughly translates into the block-level category.
        - most elements used in the body of documents are flow content (eg table, video, embed, article etc)
        - elements can wrap (almost) all other elements
    - o sectioning content
        - content that defines the scope of headings and footers (eg article, nav, section etc)
    - o heading content
        - defines the header of a section (eg h1, h2 etc)
        - they convey a meaning – the content they wrap is to be treated as heading content
    - o phrasing content
        - roughly translates into the traditional inline category
        - defines the text and the markup it contains (eg abbr, cite, em etc)
    - embedded content
        - imports another resource (eg audio, embed, iframe, img, video, object etc)
    - interactive content
        - elements that are specifically designed for user interaction (eg a, button, embed, iframe, select etc)



22 - The Semantic Web

- gives content on the web page meaning and structure by using the correct HTML element
- describes the content rather than how content should look
- enables computers, screen readers, search engines and other devices to understand the content
- always use semantic elements if they exist

Elements

- used for layout only – semantically meaningless
- <header></header>
- <footer></footer>
- <nav></nav>
- <article></article>
- <section></section>
- <hr> used to signify and thematic break in content
- emphasis
    - while both are typically (but not always) styled to display as italic, is used when you want to put more stress on a word or phrase rather than just display a word or phrase using italics. is semantic mark up
- importance
    - both will embolden a word or phrase so they will look the same in a browser, however only is of use to a screen reader. is semantic mark up.
- <aside></aside>
- <main></main>

CSS (Cascading Style Sheets)

- CSS is a language that describes the style of an HTML document.
- CSS describes how HTML elements are to be displayed on screen, paper, or in other media
- CSS saves a lot of work. It can control the layout of multiple web pages all at once
- External style sheets are stored in CSS files

Why use CSS?

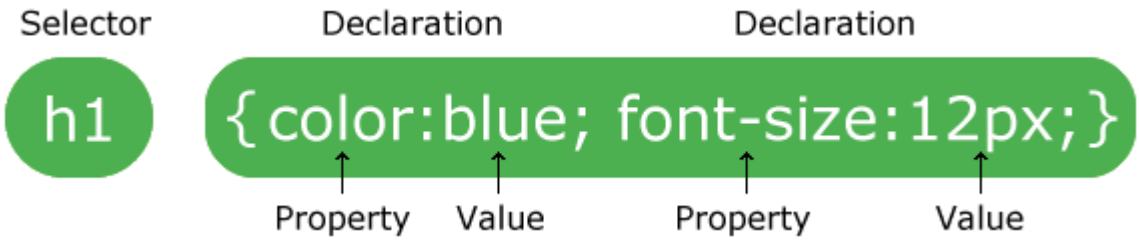
- CSS is used to define styles for your web pages, including the design, layout and variations in display for different devices and screen sizes.
- HTML was NEVER intended to contain tags for formatting a web page! HTML was created to describe the content of a web page, like:

```
<h1>This is a heading</h1>

<p>This is a paragraph.</p>
```

- When tags like <font>, and color attributes were added to the HTML 3.2 specification, it started a nightmare for web developers. Development of large websites, where fonts and color information were added to every single page, became a long and expensive process.
- To solve this problem, the World Wide Web Consortium (W3C) created CSS.
- CSS removed the style formatting from the HTML page!

CSS Syntax

- A CSS rule-set consists of a selector and a declaration block:



- The selector points to the HTML element you want to style.
- The declaration block contains one or more declarations separated by semicolons.
- Each declaration includes a CSS property name and a value, separated by a colon.
- A CSS declaration always ends with a semicolon, and declaration blocks are surrounded by curly braces.
- In the following example all <p> elements will be center-aligned, with a red text color:

CSS Selectors

CSS selectors are used to "find" (or select) HTML elements based on their element name, id, class, attribute, and more.

The element Selector

The element selector selects elements based on the element name. You can select all <p> elements on a page like this (in this case, all <p> elements will be center-aligned, with a red text color):

```
<!DOCTYPE html>
<html>
<head>
<style>
p {
    color: red;
    text-align: center;
}
</style>
</head>
<body>

<p>Hello World!</p>
<p>These paragraphs are styled with CSS.</p>

</body>
</html>
```

```
<!DOCTYPE html>
<html>
<head>
<style>
body {
    background-color: lightblue;
}

h1 {
    color: white;
    text-align: center;
}

p {
    font-family: verdana;
    font-size: 20px;
}
</style>
</head>
<body>

<h1>My First CSS Example</h1>
<p>This is a paragraph.</p>

</body>
</html>
```

The id Selector

The id selector uses the id attribute of an HTML element to select a specific element.

The id of an element should be unique within a page, so the id selector is used to select one unique element!

To select an element with a specific id, write a hash (#) character, followed by the id of the element.

The style rule below will be applied to the HTML element with id="para1":

An id name cannot start with a number.

```
<!DOCTYPE html>
<html>
<head>
<style>
#para1 {
    text-align: center;
    color: red;
}
</style>
</head>
<body>

<p id="para1">Hello World!</p>
<p>This paragraph is not affected by the style.</p>

</body>
</html>
```

The class Selector

The class selector selects elements with a specific class attribute.

To select elements with a specific class, write a period (.) character, followed by the name of the class.

In the example below, all HTML elements with class="center" will be red and center-aligned:

```
<!DOCTYPE html>
<html>
<head>
<style>
.center {
    text-align: center;
    color: red;
}
</style>
</head>
<body>

<h1 class="center">Red and center-aligned heading</h1>
<p class="center">Red and center-aligned paragraph.</p>

</body>
</html>
```

You can also specify that only specific HTML elements should be affected by a class. In the example below, only <p> elements with class="center" will be center-aligned:

```
<!DOCTYPE html>
<html>
<head>
<style>
p.center {
    text-align: center;
    color: red;
}
</style>
</head>
<body>

<h1 class="center">This heading will not be affected</h1>
<p class="center">This paragraph will be red and center-aligned.</p>

</body>
</html>
```

HTML elements can also refer to more than one class. In the example below, the <p> element will be styled according to class="center" and to class="large":

```
<p class="center large">This paragraph will be red,
 center-aligned, and in a large font-size.</p>
```

```
p.center {
    text-align: center;
    color: red;
}

p.large {
    font-size: 300%;
}
```

Grouping Selectors

If you have elements with the same style definitions, like this ->

```
h1 {
    text-align: center;
    color: red;
}

h2 {
    text-align: center;
    color: red;
}

p {
    text-align: center;
    color: red;
}
```

It will be better to group the selectors, to minimize the code. To group selectors, separate each selector with a comma.

In the example below we have grouped the selectors from the code above:

```
h1, h2, p {
    text-align: center;
    color: red;
}
```

CSS Comments

Comments are used to explain the code, and may help when you edit the source code at a later date. Comments are ignored by browsers.

A CSS comment starts with /* and ends with */. Comments can also span multiple lines:

```
p {
    color: red;
    /* This is a single-line comment */
    text-align: center;
}

/* This is
a multi-line
comment */
```

Exercises

1. Change the color of all <p> elements to "red".
2. Change the color of the element with id="para1", to "red".
3. Change the color of all elements with the class "colortext", to "red".
4. Change the color of all <p> and <h1> elements, to "red". Group the selectors to minimize code.

Ways to insert CSS

- External style sheet
- Internal style sheet
- Inline style

External Style Sheet

With an external style sheet, you can change the look of an entire website by changing just one file! Each page must include a reference to the external style sheet file inside the <link> element. The <link> element goes inside the <head> section:

```
<head>
<link rel="stylesheet" type="text/css" href="mystyle.css">
</head>
```

An external style sheet can be written in any text editor. The file should not contain any html tags. The style sheet file must be saved with a .css extension.

Here is how the "mystyle.css" looks ->

```
body {
    background-color: lightblue;
}

h1 {
    color: navy;
    margin-left: 20px;
}
```

Do not add a space between the property value and the unit (such as margin-left: **20 px**;).

The correct way is: margin-left: **20px**;

Internal Style Sheet

An internal style sheet may be used if one single page has a unique style.

Internal styles are defined within the <style> element, inside the <head> section of an HTML page:

```
<head>
<style>
body {
    background-color: linen;
}

h1 {
    color: maroon;
    margin-left: 40px;
}
</style>
</head>
```

Inline Styles

An inline style may be used to apply a unique style for a single element.

To use inline styles, add the style attribute to the relevant element. The style attribute can contain any CSS property.

The example below shows how to change the color and the left margin of a <h1> element:

```
<h1 style="color:blue;margin-left:30px;">This is a heading</h1>
```

An inline style loses many of the advantages of a style sheet (by mixing content with presentation). Use this method sparingly.

Multiple Style Sheets

If some properties have been defined for the same selector (element) in different style sheets, the value from the **last read style sheet** will be used.

```
h1 {
    color: navy;
}
```

Assume that an external style sheet has the following style for the <h1> element: set to navy blue. Then, assume that an internal style sheet also has the following style for the <h1> element to orange. If the internal style is defined after the link to the external style sheet, the <h1> elements will be "orange":

Cascading Order

What style will be used when there is more than one style specified for an HTML element?

Generally speaking we can say that all the styles will "cascade" into a new "virtual" style sheet by the following rules, where number one has the highest priority:

- Inline style (inside an HTML element)
- External and internal style sheets (in the head section)
- Browser default

So, an inline style (inside a specific HTML element) has the highest priority, which means that it will override a style defined inside the <head> tag, or in an external style sheet, or a browser default value.

The CSS `margin` properties are used to create space around elements, outside of any defined borders.

With CSS, you have full control over the margins. There are properties for setting the margin for each side of an element (top, right, bottom, and left).

Margin - Individual Sides

CSS has properties for specifying the margin for each side of an element:

- `margin-top`
- `margin-right`
- `margin-bottom`
- `margin-left`

All the margin properties can have the following values:

- auto - the browser calculates the margin
- *length* - specifies a margin in px, pt, cm, etc.
- *%* - specifies a margin in % of the width of the containing element
- inherit - specifies that the margin should be inherited from the parent element

Tip: Negative values are allowed.

```
p {
    margin-top: 100px;
    margin-bottom: 100px;
    margin-right: 150px;
    margin-left: 80px;
}
```

The auto Value

You can set the margin property to `auto` to horizontally center the element within its container. The element will then take up the specified width, and the remaining space will be split equally between the left and right margins:

```
div {
    width: 300px;
    margin: auto;
    border: 1px solid red;
}
```

The inherit Value

This example lets the left margin of the <p class="ex1"> element be inherited from the parent element ( ):

```
div {
    border: 1px solid red;
    margin-left: 100px;
}

p.ex1 {
    margin-left: inherit;
}
```

Margin Collapse

Top and bottom margins of elements are sometimes collapsed into a single margin that is equal to the largest of the two margins.

This does not happen on left and right margins! Only top and bottom margins!

Look at the following example:

```
h1 {
    margin: 0 0 50px 0;
}

h2 {
    margin: 20px 0 0 0;
}
```

CSS Height and Width

```
div {
    height: 200px;
    width: 50%;
    background-
color: powderblue;
}
```

The `height` and `width` properties are used to set the height and width of an element.

The `height` and `width` can be set to auto (this is default. Means that the browser calculates the height and width), or be specified in *length values*, like px, cm, etc., or in percent (%) of the containing block.

Note: The `height` and `width` properties do not include padding, borders, or margins; they set the height/width of the area inside the padding, border, and margin of the element!

Setting max-width

The `max-width` property is used to set the maximum width of an element.

The `max-width` can be specified in *length values*, like px, cm, etc., or in percent (%) of the containing block, or set to none (this is default. Means that there is no maximum width).

The problem with the `<div>` above occurs when the browser window is smaller than the width of the element (500px). The browser then adds a horizontal scrollbar to the page.

Using `max-width` instead, in this situation, will improve the browser's handling of small windows.

Note: The value of the `max-width` property overrides `width`.

The following example shows a `<div>` element with a height of 100 pixels and a max-width of 500 pixels:

```
div {
    max-width: 500px;
    height: 100px;
    background-color: powderblue;
}
```

CSS Padding

The CSS `padding` properties are used to generate space around an element's content, inside of any defined borders.

With CSS, you have full control over the padding. There are properties for setting the padding for each side of an element (top, right, bottom, and left).

Padding - Individual Sides

CSS has properties for specifying the padding for each side of an element:

- `padding-top` `padding-right` `padding-bottom` `padding-left`

All the padding properties can have the following values:

- *length* - specifies a padding in px, pt, cm, etc.
- *%* - specifies a padding in % of the width of the containing element
- inherit - specifies that the padding should be inherited from the parent element

Note: Negative values are not allowed.

The following example sets different padding for all four sides of a element:

```
div {
    padding-top: 50px;
    padding-right: 30px;
    padding-bottom: 50px;
    padding-left: 80px;
}
```

Padding - Shorthand Property

To shorten the code, it is possible to specify all the padding properties in one property.

The `padding` property is a shorthand property for the following individual padding properties:

`padding-top` `padding-right` `padding-bottom` `padding-left`

If the `padding` property has four values:

- **padding: 25px 50px 75px 100px;**
  - top padding is 25px
  - right padding is 50px
  - bottom padding is 75px
  - left padding is 100px

```
div {
    padding: 25px 50px 75px 100px;
}
```

If the `padding` property has three values:

- **padding: 25px 50px 75px;**
  - top padding is 25px
  - right and left paddings are 50px
  - bottom padding is 75px

If the `padding` property has two values:

- **padding: 25px 50px;**
  - top and bottom paddings are 25px
  - right and left paddings are 50px

If the `padding` property has one value:

- **padding: 25px;**
  - all four paddings are 25px

Padding and Element Width

The CSS `width` property specifies the width of the element's content area. The content area is the portion inside the padding, border, and margin of an element (the box model).

So, if an element has a specified width, the padding added to that element will be added to the total width of the element. This is often an undesirable result.

In the following example, the element is given a width of 300px. However, the actual rendered width of the element will be 350px (300px + 25px of left padding + 25px of right padding):

With the **CSS box-sizing Property**. The **box-sizing property** allows us to include the padding and border in an element's total width and height. If you set **box-sizing**: border-**box**; on an element padding and border are included in the width and height: Both divs are the same **size** now!

```
div {
    width: 300px;
    padding: 25px;
}
```

```
div {
    width: 300px;
    padding: 25px;
```

```
    box-sizing: border-box;
}
```

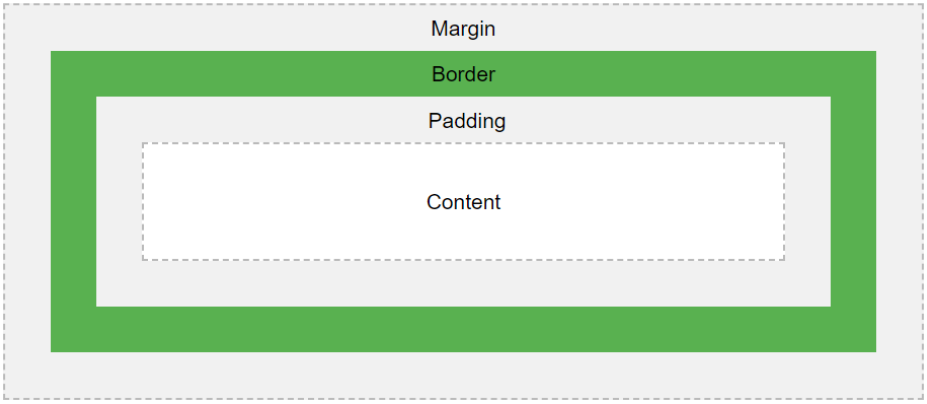
The CSS Box Model

All HTML elements can be considered as boxes. In CSS, the term "box model" is used when talking about design and layout.

The CSS box model is essentially a box that wraps around every HTML element. It consists of: margins, borders, padding, and the actual content. The image below illustrates the box model:

- **Content** - The content of the box, where text and images appear
- **Padding** - Clears an area around the content. The padding is transparent
- **Border** - A border that goes around the padding and content
- **Margin** - Clears an area outside the border. The margin is transparent

The box model allows us to add a border around elements, and to define space between elements.




```
div {
  background-color: lightgrey;
  width: 300px;
  border: 25px solid green;
  padding: 25px;
  margin: 25px;
}
```

Width and Height of an Element

In order to set the width and height of an element correctly in all browsers, you need to know how the box model works.

Important: When you set the width and height properties of an element with CSS, you just set the width and height of the **content area**. To calculate the full size of an element, you must also add padding, borders and margins.

Assume we want to style a element to have a total width of 350px:

Here is the calculation:

320px (width)
+ 20px (left + right padding)
+ 10px (left + right border)
+ 0px (left + right margin)
= 350px

```
div {
    width: 320px;
    padding: 10px;
    border: 5px solid gray;
    margin: 0;
}
```

The total width of an element should be calculated like this:

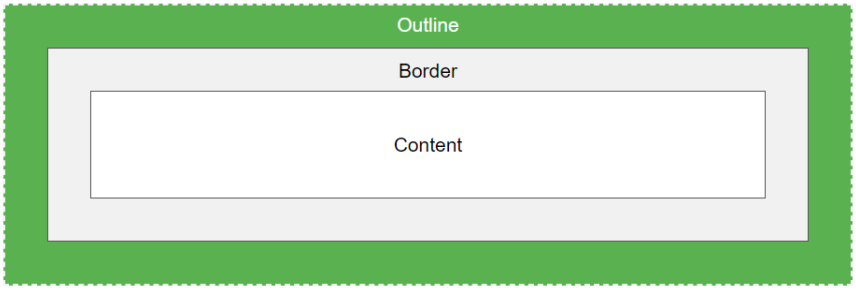
Total element width = width + left padding + right padding + left border + right border + left margin + right margin

The total height of an element should be calculated like this:

Total element height = height + top padding + bottom padding + top border + bottom border + top margin + bottom margin

CSS Outline

An outline is a line that is drawn around elements, OUTSIDE the borders, to make the element "stand out".



CSS has the following outline properties:

- `outline-style`
- `outline-color`
- `outline-width`
- `outline-offset`
- `outline`

Note: Outline differs from borders! Unlike border, the outline is drawn outside the element's border, and may overlap other content. Also, the outline is NOT a part of the element's dimensions; the element's total width and height is not affected by the width of the outlin

Outline Style

The `outline-style` property specifies the style of the outline, and can have one of the following values:

- `dotted` - Defines a dotted outline,
- `dashed` - Defines a dashed outline,
- etc.

Note: None of the other outline properties will have any effect, unless the `outline-style` property is set!

```
p {
  border: 2px solid black;
  outline: #4CAF50 solid 10px;
  margin: auto;
  padding: 10px;
  text-align: center;
}
```

```
<style> p {outline-color:red;}

        p.dotted {outline-style: dotted;}  </style>
```

web-safe fonts

- each device comes with its own pre-installed font selection based largely on its operating system
- problem is systems can differ
- if this is not taken into account then the font a user sees when they view a web page may not be the font that was intended
- web-safe fonts are fonts that will appear across all operating systems
- designers should specify fonts to fall back to if the font they use is not recognised by a particular operating system
- common web safe fonts include Arial, Helvetica, Times New Roman etc
- embedding web fonts

The 'Web safe' ones, appear across all operating systems. They're the small collection of fonts that overlap from Windows to Mac to Google.

- **Arial**. **Arial** is like the de facto standard for most. ...
- **Helvetica**. **Helvetica** is usually the designers' go-to **sans serif** font. ...
- **Times New Roman**. **Times New Roman** is to **serif** what **Arial** is to **sans serif**. ...
- **Courier New**. ..., C**ourier**. ..., **Verdana**. ...,**Georgia**.

Embedding web fonts

Font embedding has been around for years within popular offline applications, and of course, designers have been experimenting with type and unique fonts for years before with design software. Recently, we have been able to include fonts directly on web servers, and therefore include them via CSS right into our websites.

Why Use Font Embedding?

Improved Design - Using the same old, plain fonts that have been used for years previously just doesn't stand out in the same way it did before. Standard web safe fonts are still great for large blocks of content, but headers today call for cleaner fonts, more professional fonts, and better designed fonts. Plus, from a design perspective, we can have more creative freedom.

Better SEO - Search engines find content and prioritize it on a webpage based on the tags they're in. Header tags will take on much more leverage than the alt tag of an image, so it's easy to see why using font embedding for headlines, blog post titles, and other important pieces of content can be better off in a header tag, rather than used as an image.

Ease of Use - Get the font(s) of your choice set up during the development process, and enjoy lighter maintenance later. No need to create custom images for headers, sliders, or otherwise. Simply style, and type out whatever content is needed. That content can then change easily, which is great for updates on your end, and ease of use on a client's end

Activity 25: Semantic mark up

- semantic markup that adds textual meaning
    - quotations
        - <blockquote></blockquote> long section that is quoted from another source
        - <q></q> short quotation
    - abbreviations and acronyms
        - <abbr></abbr>
    - citations and definitions
        - <cite></cite>
        - <dfn></dfn>
    - address
        - <address></address>
    - mark
        - <mark></mark>
- semantic markup for self-contained content
    - figure
        - <figure></figure>
    - caption
        - <caption></caption>

10.1.3 Creating effective pages

Visual hierarchy refers to the arrangement or presentation of elements in a way that implies importance. In other words, **visual hierarchy** influences the order in which the human eye perceives what it sees. This order is created by the **visual** contrast between forms in a field of perception.

Designing Visual Hierarchy

- Size. ...
- Position. ...
- Negative space. ...
- Color. ...
- Contrast. ...
- Repetition: Repetition provides meaning to **visual** elements and allows one to more easily and quickly assimilate information. ...
- Alignment: Alignment creates order and **visual** coherence to a design

THE BASICS OF COLOR THEORY

The colors in a color wheel can be divided into three groups: primary colors (red, blue, and yellow); secondary colors, which are obtained by mixing two primary colors (orange, green, and purple); and tertiary colors, which are a combination of one primary and one secondary color (yellow-green, red-orange, etc.).

How to control color opacity with RGBA in CSS

When it comes to specifying colors in CSS, the RGB color model has enjoyed immense popularity for many years now. The designer uses a combination of red, green, and blue to achieve the required tone. ...

White or black can be added to each color to create a different tint or shade until the number of colors becomes infinite.

Color theory in web design comes down to three main things:

1. **Complementation**: How we see colors in terms of their relationship to other colors.
2. **Contrast**: How we use colors to reduce eyestrain and focus users' attention to certain elements on a page.
3. **Vibrancy**: How we use color to influence the emotions of users or attract a specific target market.

COMPLEMENTATION

Opposite colors on the color wheel – such as red and green, or purple and yellow – are complementary. They create a strong contrast, which results in a balance that can draw a user's attention and help to build a specific energy.

Split complementary, or compound color schemes, gives designers more wiggle room. Using two colors from each opposing end of the color wheel, it still provides a contrast, but one that's a little softer.

CONTRAST

Contrast is the area of color theory with the biggest impact on the usability of a website. It refers to the level of clarity between two objects on a page, most notably between text and background color. Low contrast may look beautiful and harmonious, but it's much harder to read, especially if you have a sight disability or are using a mobile device outside on a sunny day.

The safest choice is a **dark color on a light background**, or vice versa, although try to avoid using pure black – in real life, it's rarely seen and you want to reflect real-world believability.

If you're working with a light background, opt for a dark grey, or a deep blue, green, purple, etc. for your text instead.

The level of contrast on your website has a profound effect on its accessibility for people with disabilities that affect their sight. Some 8% of the population suffer from a form of color blindness. That means choosing the wrong colors, such as red and green, will make your website virtually unreadable for them. Always choose a high contrast, such as blues and yellows, and ensure that color is not the only visual cue on the website.

VIBRANCY, OR THE PSYCHOLOGY OF COLOR

Finally, the vibrancy of the colors you choose has a profound effect on your users' emotions, a concept known as the psychology of color.

Of course, sometimes your colors are dictated by the logo or wider branding material of the business or organization. Or you're targeting a particular community where certain colors hold certain associations. For example, in China red symbolizes prosperity and happiness.

What that color says about your brand

Colors are emotional. Whether you realize it or not, individual colors are capable of invoking particular feelings inside of us. And while the feelings can be influenced by personal experiences,

Warm colors: Red, orange, and yellow

- **Red**: A hot, stimulating color, symbolizing love, passion, and power. Also associated with anger and danger. Outside the Western world, it stands for prosperity and happiness in China, is the color for brides in eastern cultures, and is used to signify communism.
- **Orange**: Less overwhelming than red, orange is a cheerful color that evokes joy and happiness, and is regarded as friendly. Due to its association with the changing seasons, it can be used to symbolize movement.
- **Yellow**: The brightest color and often considered the most energizing. Associated with joy, happiness, sunshine, and intelligence, it's a positive, optimistic color. Unless you're in Egypt where it's the color of mourning.

Cool colors: Green, blue, and purple

- **Green**: Bridging the gap between warm and cool colors, green symbolizes nature and has a healing quality, therefore has a lot of positive connotations. It's also used for growth, harmony, and the environment.
- **Blue**: A very dependable color, blue signifies stability, expertise, and trust. It's a calming color, though the exact hue varies its meaning, with darker tones used for corporate website and light blues for friendly, open sites, such as social media.
- **Purple**: Traditionally the color of royalty, dark shades signify sophistication, wealth, and luxury, while the lighter hues represent spring and romance. Also used for creativity, imagination, and spirituality, it has different meanings around the world, such as Thailand where it's the traditional mourning color for widows.

Neutral colors: Black, white, grey, and brown

Neutral colors are mainly used for backgrounds and are often accented with brighter colors that provide the real impact on the page. They do, however, hold their own associations:

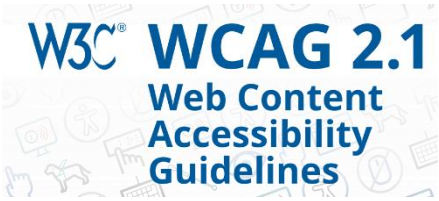
- Black: Wealth, power, elegance, sophistication, death.
- White: Purity, innocence, freshness, cleanliness.
- Grey: Tradition, calmness, conservatism, somberness.
- Brown: Earthiness, dependability, reliability.

USING COLOR IN WEB DESIGN

The colors you choose for a website give it meaning, without the need for any descriptive words. They provide an initial, instant impact for your users. So choosing the right colors is crucial to capturing the right emotions and mood, otherwise you'll see a high bounce rate as users navigate away believing the site to be untrustworthy or unprofessional.

But get your combinations right, and they'll influence the entire design, from the tone, style and emotions, to the usability of the website.

Principles of accessibility



The Web Content Accessibility Guidelines (WCAG) are part of a series of web accessibility guidelines published by the Web Accessibility Initiative (WAI) of the World Wide Web Consortium (W3C), the main international standards organization for the Internet. They are a set of recommendations for making Web content more accessible, primarily for people with disabilities—but also for all user agents, including highly limited devices, such as mobile phones. The current version, WCAG 2.0, was published in December 2008 and became an ISO standard,

WCAG 1.0 consist of 14 guidelines—each of which describes a general principle of accessible design. Each guideline covers a basic theme of web accessibility and is associated with one or more checkpoints that describes how to apply that guideline to particular webpage features.

- Guideline 1: Provide equivalent alternatives to auditory and visual content
- Guideline 2: Don't rely on colour alone
- Guideline 3: Use markup and style sheets, and do so properly
- Guideline 4: Clarify natural language usage
- Guideline 5: Create tables that transform gracefully
- Guideline 6: Ensure that pages featuring new technologies transform gracefully
- Guideline 7: Ensure user control of time sensitive content changes
- Guideline 8: Ensure direct accessibility of embedded user interfaces
- Guideline 9: Design for device independence
- Guideline 10: User interim solutions
- Guideline 11: Use W3C technologies and guidelines
- Guideline 12: Provide context and orientation information
- Guideline 13: Provide clear navigation mechanisms
- Guideline 14: Ensure that documents are clear and simple

Principles of usability

Usability - when you can conveniently use a product or service and it meets your expectations.

"The 10 most general principles for interaction design. They are called 'heuristics' because they are more in the nature of rules of thumb than specific usability guidelines."

- Visibility of system status - The system should always keep users informed about what is going on, through appropriate feedback within reasonable time.
- Match between system and the real world - The system should speak the users' language, with words, phrases and concepts familiar to the user, rather than system-oriented terms.
- User control and freedom - Users often choose system functions by mistake and will need a clearly marked "emergency exit"
- Consistency and standards - Users should not have to wonder whether different words, situations, or actions mean the same thing.
- Error prevention - Even better than good error messages is a careful design which prevents a problem from occurring in the first place.
- Recognition rather than recall - Minimize the user's memory load by making objects, actions, and options visible. Instructions for use of the system should be visible or easily retrievable whenever appropriate.
- Dialogues should not contain information which is irrelevant or rarely needed.
- Help and documentation

A “breadcrumb” navigation reveals the user’s location in a website. On websites that have a lot of pages, breadcrumb navigation can greatly enhance the way users find their way around. In terms of usability, breadcrumbs reduce the number of actions a website visitor needs to take in order to get to a higher-level page, and they improve the findability of website sections and pages.

Other menu systems include horizontal scroll, vertical, drop down and button groups.

Cross browser compatible - The ability for a website, web application, HTML construct or client-side script to support all the web browsers.

Multi-browser means a website will work in several web browsers like Safari, Chrome, Firefox, Internet Explorer. While cross-browser means a website works in any browser, and any version of the browser, being used. This can be a struggle for developers throughout the creation of their websites. **Something that makes a website great is when it is compatible in multiple browsers regardless of version. Technically this is referred to as cross browser compatibility.**

If a user cannot view your website properly, he will not blame the operating system or his browser…he will blame the website itself. The goal as developers is to avoid that. That's why there are tools available for free, or purchase, to make your cross-browser checking easier. Examples of software include browser shots, IE tester, sauce labs.

Functional testing is carried out to ensure that the product behaves according to the functional requirements without taking design principles into consideration. This entails a series of tests which perform a feature-by-feature validation of behaviour, using a wide range of normal and erroneous input data.

In contrast, **usability testing** focuses on customer acceptance and how well the customer can use the product to complete the required task. Usability testing investigates all aspects of the usability of a product, including overall structure, navigational flow, and layout of elements on a page, clarity of content, and overall behaviour

Code validation is the process of checking that the coding of a web page is in compliance with the standards and recommendations set by the World Wide Web Consortium (W3C) for the web. **Code validation** helps to produce clean **code**.

Web development tools (often called devtools) allow web developers to test and debug their code. They are different from website builders and integrated development environments (IDEs) in that they do not assist in the direct creation of a webpage, rather they are tools used for testing the user interface of a website or web application.

Web development tools come as browser add-ons or built-in features in web browsers.

How do I open browser developer tools?
The JavaScript debugger

1. Firefox: Select ➤ Web Developer ➤ Debugger or press Ctrl + Shift + S to open the JavaScript Debugger. ...
2. Chrome: Open the Developer tools and then select the Sources tab. ...
3. Edge and Internet Explorer 11: Press F12 and then, Ctrl + 3 , or if the tools are already displayed, click on the Debugger tab.

Activity 30: Accessibility

- Web accessibility is the inclusive practice of ensuring there are no barriers that prevent interaction with, or access to, websites on the World Wide Web by people with disabilities
- All users should have equal access to information and functionality
- Web accessibility aims to include
    - visual impairments including blindness, various common types of low vision and poor eyesight and various types of colour blindness
    - mobility issues eg difficulty or inability to use hands
    - auditory (hearing) issues eg deaf or hard of hearing
    - seizures eg people who suffer from epileptic seizures caused by flashing effects etc
    - cognitive and intellectual issues eg developmental or learning difficulties etc
- WCAG
    - guidelines
        - perceivable – information cannot be invisible to all of a user's senses
        - operable – interface cannot require interaction that a user cannot perform
        - understandable – content or operation cannot be beyond their understanding
        - robust – interpreted reliably by a wide variety of user agents, including assistive technologies
- WAI-ARIA
    - technical specification published by W3C that specifies how to increase the accessibility of web pages for assistive technologies such as screen readers
    - set of attributes for plugging the accessibility gaps in HTML5 semantics
    - has no effect on how page elements are displayed or behave in browsers
    - only use where HTML5 semantics are not available or not enough
    - functionality through roles – enables the classification of otherwise meaningless tags
    - states and properties – information on how to interact with a particular widget
    - live regions for dynamic content – allow notifications whenever there are changes in that particular part of the page
    - enhanced keyboard navigation – allows every HTML element to receive keyboard focus
    - things to think about
        - images
        - audio and video
        - colours
        - text
        - links
        - forms
        - navigation and site structure
          (see checklist 1 below)
        - page regions
        - labelling regions
        - headings
        - content structure
          (see checklist 2 below)
    - tabindex, data, hidden
        - tabindex – can be used to set the focus on elements in a particular order when the user uses the tab key. tabindex=0 is default order, -1 means no tab stop. Only use them where absolutely necessary

        - eg custom elements that would not ordinarily receive focus from tab
        - data – used to store custom data (data-*)
        - hidden – Boolean attribute, when present specifies whether an element should be seen/heard. Useful for hiding elements that those with disabilities do not need to see or hear eg asterisks for required fields, those with visual impairments would not need to see those
- SEO
    - name given to activity that attempts to improve search engine rankings
    - should ensure a website can be found in search engine for words and phrases relevant to what the site is offering
    - some of most important techniques to use
        - remove anything that slows down the site
        - link to other websites with relevant content
        - write for humans first, search engines second
        - encourage other trustworthy sites to link to you
        - use web analytics to see what is working and what is not
        - write unique and relevant meta descriptions for every page
        - use readable and meaningful URLs only
        - use the right keywords in your images
- metadata
    - data about the HTML document eg
        - meta name="description" content="Free CSS Lessons">
        - meta name="keywords" content="HTML, CSS">

Tutors to provide tasks that will enable the students to

- explain/describe/discuss web accessibility
- explain/describe/discuss WCAG
- explain/describe/discuss HTML5 semantics
- explain/describe/discuss WAI-ARIA
- explain/describe/discuss SEO (including metadata)
- use metadata
- assess how well particular web pages incorporate accessibility through the use of semantics, WCAG and WAI-ARIA
- analyse the use of semantics, WCAG and WAI-ARIA employed on particular pages and make recommendations where appropriate
- assess how well particular web pages incorporate SEO techniques
- analyse the use of SEO techniques used on a particular web page and recommend improvements

- http://heydonworks.com/practical_aria_examples/ - examples of ARIA in action
- https://developer.mozilla.org/en-US/docs/Web/HTML/Global_attributes/tabindex - tab index
- https://www.w3schools.com/tags/att_global_data.asp - data
- https://www.w3schools.com/tags/att_hidden.asp - hidden
- https://www.youtube.com/watch?v=B4IqW-5a16o – SEO
- https://www.youtube.com/watch?v=sd0ypO9MTWY – SEO

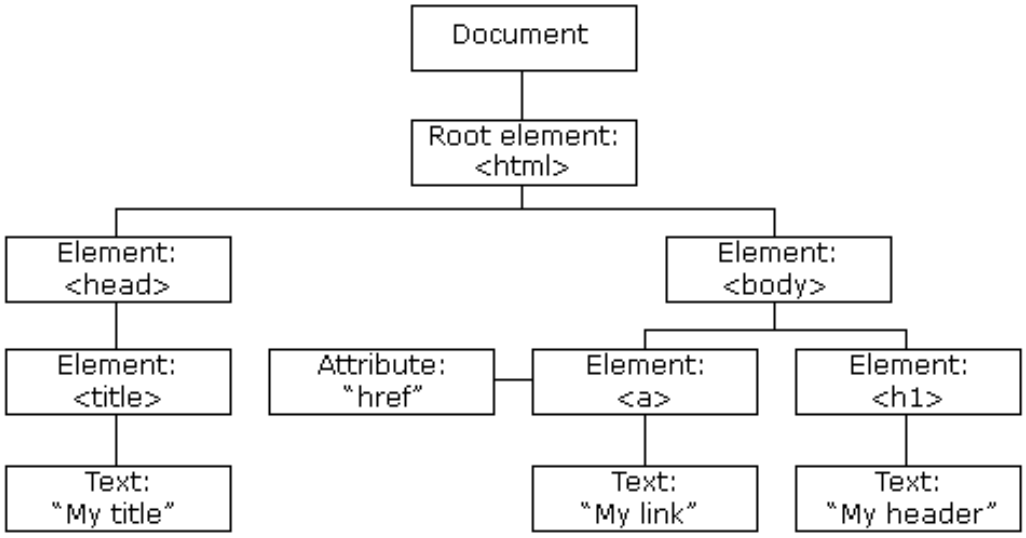
JavaScript

JavaScript is the programming language of HTML and the Web. JavaScript is one of the 3 languages all web developers must learn:

1. HTML to define the content of web pages
2. CSS to specify the layout of web pages
3. JavaScript to program the behavior of web pages

The HTML DOM (Document Object Model)

When a web page is loaded, the browser creates a **D**ocument **O**bject **M**odel of the page.



With the object model, JavaScript gets all the power it needs to create dynamic HTML:

- JavaScript can change all the HTML elements in the page
- JavaScript can change all the HTML attributes in the page
- JavaScript can change all the CSS styles in the page
- JavaScript can remove existing HTML elements and attributes
- JavaScript can add new HTML elements and attributes
- JavaScript can react to all existing HTML events in the page
- JavaScript can create new HTML events in the page

The HTML DOM is a standard for how to get, change, add, or delete HTML elements.

In the DOM, all HTML elements are defined as **objects**. The programming interface is the properties and methods of each object.

A **property** is a value that you can get or set (like changing the content of an HTML element).

A **method** is an action you can do (like add or deleting an HTML element).

Example ------->

```
<html>
<body>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML
 = "Hello World!";
</script>

</body>
</html>
```

The following example changes the content (the `innerHTML`) of the `<p>` element with `id="demo"`
In the example `getElementById` is a **method** while `innerHTML` is a **property**.

The getElementById Method

The most common way to access an HTML element is to use the `id` of the element. In the example above the `getElementById` method used `id="demo"` to find the element.

The innerHTML Property

The easiest way to get the content of an element is by using the `innerHTML` property. The `innerHTML` property is useful for getting or replacing the content of HTML elements.

JavaScript Can Change HTML Content

One of many JavaScript HTML methods is **getElementById()**.

This example uses the method to "find" an HTML element (with id="demo") and changes the element content (**innerHTML**) to "Hello JavaScript":

```
<!DOCTYPE html>
<html>
<body>
<h2>What Can JavaScript Do?</h2>
<p id="demo">JavaScript can change HTML content.</p>
<button type="button" onclick='document.getElementById("demo").innerHTML = "Hello JavaScript!"'>Click Me!</button>
</body>
</html>
```

Loading an image on the click event:

```
<button onclick="document.getElementById('myImage').src='pic_bulbon.gif'">
Turn on the light</button>
```

JavaScript Can Change HTML Styles (CSS)

Changing the style of an HTML element, is a variant of changing an HTML attribute:

```
<button type="button"
onclick="document.getElementById('demo').style.fontSize='35px'">Click Me!</button>
```

Hiding an element

```
<button type="button"
onclick="document.getElementById('demo').style.display='none'">Click Me!</button>
```

```
<p>JavaScript can show hidden HTML elements.</p>
<p id="demo" style="display:none">Hello JavaScript!</p>
<button type="button"
onclick="document.getElementById('demo').style.display='block'">Click Me!</button
```

Finding HTML Elements

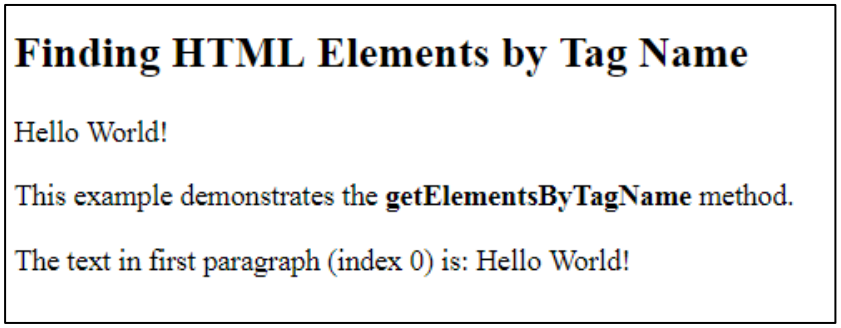
To manipulate HTML elements you have to find the elements first. There are a couple of ways to do this:

- Finding HTML elements by id (shown in the e.g. above)
- Finding HTML elements by tag name
- Finding HTML elements by class name

Finding HTML Elements by Tag Name

Exercise 17a

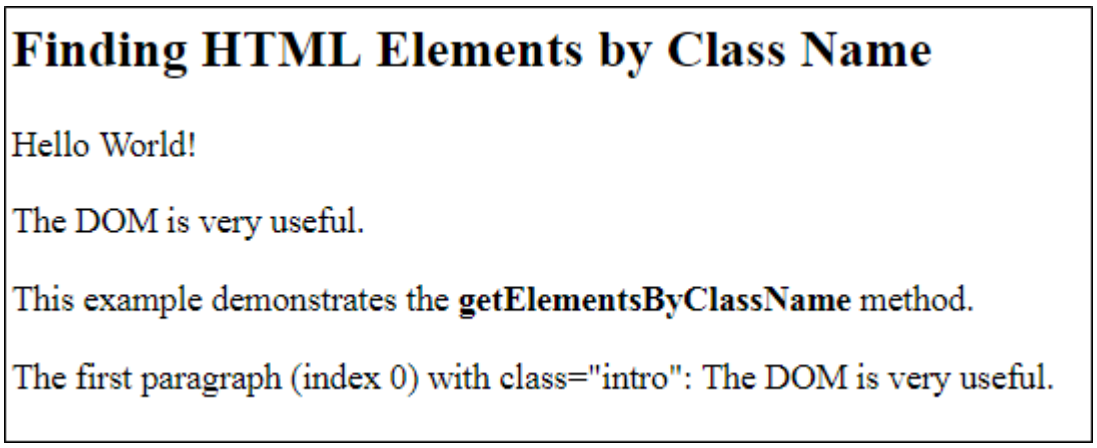
```
<!DOCTYPE html>
<html>
<body>

<h2>Finding HTML Elements by Tag Name</h2>

<p>Hello World!</p>
<p>This example demonstrates the <b>getElementsByTagName</b> method.
</p>

<p id="demo"></p>

<script>
var x = document.getElementsByTagName("p");
document.getElementById("demo").innerHTML =
'The text in first paragraph (index 0) is: ' + x[0].innerHTML;
</script>

</body>
</html>
```


Finding HTML Elements by Tag Name
Hello World!
This example demonstrates the getElementsByTagName method.
The text in first paragraph (index 0) is: Hello World!

Finding HTML Elements by Class Name

Exercise 18a

```
<!DOCTYPE html>
<html>
<body>

<h2>Finding HTML Elements by Class Name</h2>

<p>Hello World!</p>

<p class="intro">The DOM is very useful.</p>
<p class="intro">This example demonstrates the
<b>getElementsByClassName</b> method.</p>

<p id="demo"></p>

<script>
var x = document.getElementsByClassName("intro");
document.getElementById("demo").innerHTML =
'The first paragraph (index 0) with class="intro": ' + x[0].innerHTML;
</script>

</body>
</html>
```


Finding HTML Elements by Class Name
Hello World!
The DOM is very useful.
This example demonstrates the getElementsByClassName method.
The first paragraph (index 0) with class="intro": The DOM is very useful.

The <script> Tag

In HTML, JavaScript code must be inserted between <script> and </script> tags.

```
<script>
document.getElementById("demo").innerHTML = "My First JavaScript";
</script
```

Functions and Events

A JavaScript **function** is a block of JavaScript code that can be executed when "called" for. For example, a function can be called when an **event** occurs, like when the user clicks a button.

JavaScript in <head> or <body>

Scripts can be placed in the <body>, or in the <head> section of an HTML page, or in both.

In this example, a JavaScript function is placed in the <head> section of an HTML page. The function is invoked (called) when a button is clicked:

```
<!DOCTYPE html>
<html>

<head>
<script>
function myFunction() {
  document.getElementById("demo").innerHTML = "Paragraph changed.";
}
</script>
</head>
<body>

<h1>A Web Page</h1>
<p id="demo">A Paragraph</p>
<button type="button" onclick="myFunction()">Try it</button>

</body>
</html>
```

External JavaScript

Scripts can also be placed in external files:

External file: myScript.js

```
function myFunction() {
 document.getElementById("demo").innerHTML = "Paragraph changed.";
}
```

External scripts are practical when the same code is used in many different web pages. JavaScript files have the file extension **.js**.

To use an external script, put the name of the script file in the src (source) attribute of a <script> tag:

```
<script src="myScript.js"></script>
```

You can place an external script reference in <head> or <body> as you like.

The script will behave as if it was located exactly where the <script> tag is located.

External JavaScript Advantages

Placing scripts in external files has some advantages:

- It separates HTML and code
- It makes HTML and JavaScript easier to read and maintain
- Cached JavaScript files can speed up page loads

To add several script files to one page - use several script tags:

```
<script src="myScript1.js"></script>
<script src="myScript2.js"></script>
```

External References

External scripts can be referenced with a full URL or with a path relative to the current web page.

This example uses a full URL to link to a script:

```
<script src="https://www.w3schools.com/js/myScript1.js"></script>
```

This example uses a script located in a specified folder on the current web site:

```
<script src="/js/myScript1.js"></script>
```

This example links to a script located in the same folder as the current page:

```
<script src="myScript1.js"></script>
```

JavaScript Display Possibilities

JavaScript can "display" data in different ways:

- Writing into an HTML element, using **innerHTML**.
- Writing into the HTML output using **document.write()**.
- Writing into an alert box, using **window.alert()**.
- Writing into the browser console, using **console.log()**.

<script>

document.getElementById("demo").innerHTML = 15 + 6;

</script>

Using document.write()

For testing purposes, it is convenient to use **document.write()**:

```
<script>
document.write(5 + 6);
</script>
```

Using document.write() after an HTML document is loaded, will **delete all existing HTML**:

Using window.alert()

You can use an alert box to display data:

```
<script>
window.alert(5 + 6);
</script>
```

Using console.log()

For debugging purposes, you can use the **console.log()** method to display data

```
<script>
console.log(5 + 6);
</script>
```

JavaScript Statements

JavaScript Programs

A **computer program** is a list of "instructions" to be "executed" by a computer.

In a programming language, these programming instructions are called **statements**.

A **JavaScript program** is a list of programming **statements**.

In HTML, JavaScript programs are executed by the web browser.

JavaScript Statements

JavaScript statements are composed of:

Values, Operators, Expressions, Keywords, and Comments.

Semicolons ;

Semicolons separate JavaScript statements.

Add a semicolon at the end of each executable statement:

```
var a, b, c;     // Declare 3 variables
a = 5;           // Assign the value 5 to a
b = 6;           // Assign the value 6 to b
c = a + b;       // Assign the sum of a and b to c
```

When separated by semicolons, multiple statements on one line are allowed:

```
a = 5; b = 6; c = a + b;
```

JavaScript White Space

JavaScript ignores multiple spaces. You can add white space to your script to make it more readable.

The following lines are equivalent:

```
var person = "Hege";
var person="Hege";
```

A good practice is to put spaces around operators (= + - * /):

JavaScript Line Length and Line Breaks

For best readability, programmers often like to avoid code lines longer than 80 characters.

If a JavaScript statement does not fit on one line, the best place to break it is after an operator:

JavaScript Code Blocks

JavaScript statements can be grouped together in code blocks, inside curly brackets {...}.

The purpose of code blocks is to define statements to be executed together.

One place you will find statements grouped together in blocks, is in JavaScript functions:

JavaScript Syntax

JavaScript syntax is the set of rules, how JavaScript programs are constructed:

```
var x, y;           // How to declare variables
x = 5; y = 6;       // How to assign values
z = x + y;          // How to compute values
```

JavaScript Values

The JavaScript syntax defines two types of values: Fixed values and variable values.

Fixed values are called **literals**. Variable values are called **variables**.

JavaScript Literals

The most important rules for writing fixed values are:

Numbers are written with or without decimals:

```
10.50
```

```
1001
```

Strings are text, written within double or single quotes:

```
"John Doe"
```

```
'John Doe'
```

JavaScript Expressions

An expression is a combination of values, variables, and operators, which computes to a value.

The computation is called an evaluation.

For example, 5 * 10 evaluates to 50:

```
5 * 10
```

The values can be of various types, such as numbers and strings.

For example, "John" + " " + "Doe", evaluates to "John Doe"

JavaScript Keywords

JavaScript **keywords** are used to identify actions to be performed.

The **var** keyword tells the browser to create variables:

JavaScript Comments

Not all JavaScript statements are "executed".

Code after double slashes **//** or between **/*** and ***/** is treated as a **comment**.

Comments are ignored, and will not be executed:

```
var x = 5;   // I will be executed

// var x = 6;   I will NOT be executed
```

JavaScript Identifiers

Identifiers are names.

In JavaScript, identifiers are used to name variables (and keywords, and functions, and labels).

The rules for legal names are much the same in most programming languages.

In JavaScript, the first character must be a letter, or an underscore (_), or a dollar sign ($).

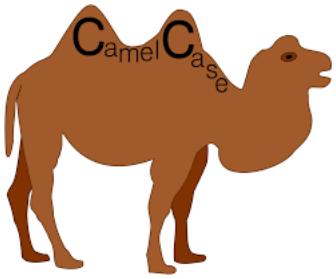
Subsequent characters may be letters, digits, underscores, or dollar signs.

Numbers are not allowed as the first character.
This way JavaScript can easily distinguish identifiers from numbers.

All JavaScript identifiers are **case sensitive**.

The variables **lastName** and **lastname**, are two different variables.

JavaScript and Camel Case



Historically, programmers have used different ways of joining multiple words into one variable name:

Hyphens:

first-name, last-name, master-card, inter-city.

Hyphens are not allowed in JavaScript. They are reserved for subtractions.

Underscore:

first_name, last_name, master_card, inter_city.

Upper Camel Case (Pascal Case):

FirstName, LastName, MasterCard, InterCity.

JavaScript Character Set

JavaScript uses the **Unicode** character set.

JavaScript Variables

JavaScript variables are containers for storing data values.

```
var x = 5;
```

In programming, just like in algebra, we use variables (like price1) to hold values.

In programming, just like in algebra, we use variables in expressions (total = price1 + price2).

From the example above, you can calculate the total to be 11.

JavaScript variables are containers for storing data values.

JavaScript Identifiers

All JavaScript **variables** must be **identified** with **unique names**.

These unique names are called **identifiers**.

Identifiers can be short names (like x and y) or more descriptive names (age, sum, totalVolume).

The general rules for constructing names for variables (unique identifiers) are:

- Names can contain letters, digits, underscores, and dollar signs.
- Names must begin with a letter
- Names can also begin with $ and _ (but we will not use it in this tutorial)
- Names are case sensitive (y and Y are different variables)
- Reserved words (like JavaScript keywords) cannot be used as names

JavaScript identifiers are case-sensitive.

The Assignment Operator

In JavaScript, the equal sign (=) is an "assignment" operator, not an "equal to" operator.

This is different from algebra. The following does not make sense in algebra:

```
x = x + 5
```

In JavaScript, however, it makes perfect sense: it assigns the value of x + 5 to x.

(It calculates the value of x + 5 and puts the result into x. The value of x is incremented by 5.)

The "equal to" operator is written like == in JavaScript.

JavaScript Data Types

JavaScript variables can hold numbers like 100 and text values like "John Doe".

In programming, text values are called text strings.

JavaScript can handle many types of data, but for now, just think of numbers and strings.

Strings are written inside double or single quotes. Numbers are written without quotes.

If you put a number in quotes, it will be treated as a text string.

Example

```
var pi = 3.14;
var person = "John Doe";
var answer = 'Yes I am!';
```

Declaring (Creating) JavaScript Variables

Creating a variable in JavaScript is called "declaring" a variable. You declare a JavaScript variable with the **var** keyword:

```
var carName;
```

After the declaration, the variable has no value. (Technically it has the value of **undefined**). To **assign** a value to the variable, use the equal sign:

```
carName = "Volvo";
```

Re-Declaring JavaScript Variables

If you re-declare a JavaScript variable, it will not lose its value. The variable carName will still have the value "Volvo" after the execution of these statements:

```
var carName = "Volvo";
var carName;
```

JavaScript Arithmetic Operators

Arithmetic operators are used to perform arithmetic on numbers: Assignment operators assign values to JavaScript variables.

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
**	Exponentiation (ES6)
/	Division
%	Modulus (Division Remainder)
++	Increment
--	Decrement

The **addition assignment** operator (+=) adds a value to a variable.

```
var x = 10;
x += 5;
```

Operator	Example	Same As
=	x = y	x = y
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y

JavaScript String Operators

The + operator can also be used to add (concatenate) strings.

```
var txt1 = "John";      var txt2 = "Doe";
var txt3 = txt1 + " " + txt2;
```

The result of txt3 will be: `John Doe`

JavaScript Comparison Operators

Operator	Description
==	equal to
===	equal value and equal type
!=	not equal
!==	not equal value or not equal type
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
?	ternary operator

JavaScript Data Types - JavaScript variables can hold many **data types**: numbers, strings, objects and more:

JavaScript Booleans - Booleans can only have two values: true or false. Booleans are often used in conditional testing.

```
var x = 5;  var y = 5;`var z = 6;
(x == y)       // Returns true
(x == z)       // Returns false
```

JavaScript Functions

A JavaScript function is a block of code designed to perform a particular task.

A JavaScript function is executed when "something" invokes it (calls it).

```
function myFunction(p1, p2) {
  return p1 * p2;    // The function returns the product of p1 and p2
}
```

JavaScript Function Syntax

A JavaScript function is defined with the **function** keyword, followed by a **name**, followed by parentheses **()**.

Function names can contain letters, digits, underscores, and dollar signs (same rules as variables).

The parentheses may include parameter names separated by commas:
(*parameter1, parameter2, ...*)

The code to be executed, by the function, is placed inside curly brackets: **{}**

```
function name(parameter1, parameter2, parameter3) {
  // code to be executed
}
```

Function **parameters** are listed inside the parentheses () in the function definition.

Function **arguments** are the **values** received by the function when it is invoked.

Inside the function, the arguments (the parameters) behave as local variables.

A Function is much the same as a Procedure or a Subroutine, in other programming languages.

Function Invocation

The code inside the function will execute when "something" **invokes** (calls) the function:

- When an event occurs (when a user clicks a button)
- When it is invoked (called) from JavaScript code
- Automatically (self invoked)

Function Invocation

The code inside the function will execute when "something" **invokes** (calls) the function:

- When an event occurs (when a user clicks a button)
- When it is invoked (called) from JavaScript code
- Automatically (self invoked)

Why Functions?

You can reuse code: Define the code once, and use it many times.

You can use the same code many times with different arguments, to produce different results.

Example

Convert Fahrenheit to Celsius:

```
function toCelsius(fahrenheit) {
  return (5/9) * (fahrenheit-32);
}
document.getElementById("demo").innerHTML = toCelsius(77);
```

The () Operator Invokes the Function

Using the example above, toCelsius refers to the function object, and toCelsius() refers to the function result.

Accessing a function without () will return the function definition instead of the function result:

Example

```
function toCelsius(fahrenheit) {
  return (5/9) * (fahrenheit-32);
}
document.getElementById("demo").innerHTML = toCelsius;
```

Local Variables

Variables declared within a JavaScript function, become **LOCAL** to the function.

Local variables can only be accessed from within the function.

Example

```
// code here can NOT use carName

function myFunction() {
  var carName = "Volvo";
  // code here CAN use carName
}

// code here can NOT use carName
```

Since local variables are only recognized inside their functions, variables with the same name can be used in different functions.

Local variables are created when a function starts, and deleted when the function is completed.

JavaScript Objects

Properties	Methods
car.name = Fiat	car.start()
car.model = 500	car.drive()
car.weight = 850kg	car.brake()
car.color = white	car.stop()

In real life, a car is an object**.**

A car has **properties** like weight and color, and **methods** like start and stop: All cars have the same **properties**, but the property **values** differ from car to car. All cars have the same **methods**, but the methods are performed **at different times**.

JavaScript Objects

You have already learned that JavaScript variables are containers for data values.

This code assigns a **simple value** (Fiat) to a **variable** named car:

```
var car = "Fiat";
```

Objects are variables too. But objects can contain many values.

This code assigns **many values** (Fiat, 500, white) to a **variable** named car:

```
var car = {type:"Fiat", model:"500", color:"white"};
```

The values are written as **name:value** pairs (name and value separated by a colon).

JavaScript objects are containers for **named values** called properties or methods.

Object Definition

You define (and create) a JavaScript object with an object literal:

```
var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
```

Spaces and line breaks are not important. An object definition can span multiple lines:

```
var person = {
  firstName:"John",
  lastName:"Doe",
  age:50,
  eyeColor:"blue"
};
```

Object Properties

The **name:values** pairs in JavaScript objects are called **properties**:

Property	Property Value
firstName	John
lastName	Doe
age	50
eyeColor	blue

Accessing Object Properties

You can access object properties in two ways:

objectName.propertyName or *objectName["propertyName"]*

```
person.lastName;
```

or

```
person["lastName"];
```

Object Methods

Objects can also have **methods**.

Methods are **actions** that can be performed on objects.

Methods are stored in properties as **function definitions**.

A method is a function stored as a property.

```
var person = {
  firstName: "John",
  lastName : "Doe",
  id       : 5566,
  fullName : function() {
    return this.firstName + " " + this.lastName;
  }
};
```

The **this** Keyword

In a function definition, **this** refers to the "owner" of the function.

In the example above, **this** is the **person object** that "owns" the **fullName** function.

In other words, **this.firstName** means the **firstName** property of **this object**.

Accessing Object Methods

You access an object method with the following syntax:

objectName.methodName()

```
name = person.fullName();
```

If you access a method **without** the () parentheses, it will return the **function definition**:

```
name = person.fullName;
```

JavaScript Events

HTML events are **"things"** that happen to HTML elements.

When JavaScript is used in HTML pages, JavaScript can **"react"** on these events.

HTML Events

An HTML event can be something the browser does, or something a user does.

Here are some examples of HTML events:

- An HTML web page has finished loading
- An HTML input field was changed
- An HTML button was clicked

Often, when events happen, you may want to do something.

JavaScript lets you execute code when events are detected.

HTML allows event handler attributes, **with JavaScript code**, to be added to HTML elements.

With single quotes:

```
<element event='some JavaScript'>
```

With double quotes:

```
<element event="some JavaScript">
```

In the following example, an onclick attribute (with code), is added to a button element:

Example

```
<button onclick="document.getElementById('demo').innerHTML = Date()">The time is?</button>
```

In the example above, the JavaScript code changes the content of the element with id="demo".

In the next example, the code changes the content of its own element (using **this**.innerHTML):

Example

```
<button onclick="this.innerHTML = Date()">The time is?</button>
```

JavaScript code is often several lines long. It is more common to see event attributes calling functions:

Example

```
<button onclick="displayDate()">The time is?</button>
```

Common HTML Events

Event	Description
onchange	An HTML element has been changed
onclick	The user clicks an HTML element
onmouseover	The user moves the mouse over an HTML element
onmouseout	The user moves the mouse away from an HTML element
onkeydown	The user pushes a keyboard key
onload	The browser has finished loading the page

What can JavaScript Do?

Event handlers can be used to handle, and verify, user input, user actions, and browser actions:

- Things that should be done every time a page loads
- Things that should be done when the page is closed
- Action that should be performed when a user clicks a button
- Content that should be verified when a user inputs data
- And more ...

Many different methods can be used to let JavaScript work with events:

- HTML event attributes can execute JavaScript code directly
- HTML event attributes can call JavaScript functions
- You can assign your own event handler functions to HTML elements
- You can prevent events from being sent or being handled
- And more ...

Special Characters

Because strings must be written within quotes, JavaScript will misunderstand this string:

```
var x = "We are the so-called "Vikings" from the north.";
```

The string will be chopped to "We are the so-called ".

The solution to avoid this problem, is to use the **backslash escape character**.

The backslash (\) escape character turns special characters into string characters:

JavaScript String Methods

String methods help you to work with strings.

String Methods and Properties

Primitive values, like "John Doe", cannot have properties or methods (because they are not objects). But with JavaScript, methods and properties are also available to primitive values, because JavaScript treats primitive values as objects when executing methods and properties.

String Length

The **length** property returns the length of a string:

```
var txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
var sln = txt.length;
```

Finding a String in a String

The indexOf() **method returns the index of (the position of)** the **first** occurrence of a specified text in a string:

```
var str = "Please locate where 'locate' occurs!";
var pos = str.indexOf("locate");
```

JavaScript counts positions from zero. 0 is the first position in a string, 1 is the second, 2 is the third ...

The **lastIndexOf()** method returns the index of the **last** occurrence of a specified text in a string:

```
var str = "Please locate where 'locate' occurs!";
var pos = str.lastIndexOf("locate");
```

Both indexOf(), and lastIndexOf() return -1 if the text is not found.

```
var str = "Please locate where 'locate' occurs!";
var pos = str.lastIndexOf("John");
```

Both methods accept a second parameter as the starting position for the search:

```
var str = "Please locate where 'locate' occurs!";
var pos = str.indexOf("locate",15);
```

Searching for a String in a String

The **search()** method searches a string for a specified value and returns the position of the match:

```
var str = "Please locate where 'locate' occurs!";
var pos = str.search("locate");
```

The two methods, indexOf() and search(), are **equal?**

They accept the same arguments (parameters), and return the same value?

The two methods are **NOT** equal. These are the differences:

- The search() method cannot take a second start position argument.
- The indexOf() method cannot take powerful search values (regular expressions).

JavaScript if else and else if

In JavaScript we have the following conditional statements:

- Use `if` to specify a block of code to be executed, if a specified condition is true
- Use `else` to specify a block of code to be executed, if the same condition is false
- Use `else if` to specify a new condition to test, if the first condition is false
- Use `switch` to specify many alternative blocks of code to be executed

```
if (hour < 18) {
  greeting = "Good day";
}
```

```
if (hour < 18) {
  greeting = "Good day";
} else {
  greeting = "Good evening";
}
```

```
if (time < 10) {
  greeting = "Good morning";
} else if (time < 20) {
  greeting = "Good day";
} else {
  greeting = "Good evening";
}
```

The JavaScript Switch Statement

Use the `switch` statement to select one of many code blocks to be executed.

Syntax

This is how it works:

- The switch expression is evaluated once.
- The value of the expression is compared with the values of each case.
- If there is a match, the associated block of code is executed.

```
switch(expression) {
  case x:
    // code block
    break;
  case y:
    // code block
    break;
  default:
    // code block
}
```

The break Keyword

When JavaScript reaches a `break` keyword, it breaks out of the switch block. This will stop the execution of more code and case testing inside the block.

A break can save a lot of execution time because it "ignores" the execution of all the rest of the code in the switch block.

```
switch (new Date().getDay()) {
  case 6:
    text = "Today is Saturday";
    break;
  case 0:
    text = "Today is Sunday";
    break;
  default:
    text = "Looking forward to the Weekend";
}
```

Common Code Blocks

Sometimes you will want different switch cases to use the same code.

In this example case 4 and 5 share the same code block, and 0 and 6 share another code block:

```
switch (new Date().getDay()) {
  case 4:
  case 5:
    text = "Soon it is Weekend";
    break;
  case 0:
  case 6:
    text = "It is Weekend";
    break;
  default:
    text = "Looking forward to the Weekend";
}
```

JavaScript Loops

Loops are handy, if you want to run the same code over and over again, each time with a different value.

Often this is the case when working with arrays:

Instead of writing:

```
text += cars[0] + "<br>";
text += cars[1] + "<br>";
text += cars[2] + "<br>";
text += cars[3] + "<br>";
text += cars[4] + "<br>";
text += cars[5] + "<br>";
```

You can write:

```
var i;
for (i = 0; i < cars.length; i++) {
  text += cars[i] + "<br>";
}
```

Different Kinds of Loops

JavaScript supports different kinds of loops:

- `for` - loops through a block of code a number of times
- `for/in` - loops through the properties of an object
- `while` - loops through a block of code while a specified condition is true
- `do/while` - also loops through a block of code while a specified condition is true

From the example above, you can read:

```
for (i = 0; i < 5; i++) {
  text += "The number is
" + i + "<br>";
}
```

Statement 1 sets a variable before the loop starts (var i = 0).

Statement 2 defines the condition for the loop to run (i must be less than 5).

Statement 3 increases a value (i++) each time the code block in the loop has been executed.

The For/In Loop

The JavaScript `for/in` statement loops through the properties of an object:

```
var person = {fname:"John", lname:"Doe", age:25};

var text = "";
var x;
for (x in person) {
  text += person[x];
}
```

The While Loop

The `while` loop loops through a block of code as long as a specified condition is true.

```
while (i < 10) {
  text += "The number is " + i;
  i++;
}
```

The Do/While Loop

The `do/while` loop is a variant of the while loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

```
do {
  text += "The number is " + i;
  i++;
}
while (i < 10);
```

Do not forget to increase the variable used in the condition, otherwise the loop will never end!

Converting Numbers to Strings

The global method `String()` can convert numbers to strings.

It can be used on any type of numbers, literals, variables, or expression

```
String(x)         // returns a string from a number variable x
String(123)       // returns a string from a number literal 123
String(100 + 23)  // returns a string from a number from an expression
```

Converting Strings to Numbers

The global method `Number()` can convert strings to numbers.

Strings containing numbers (like "3.14") convert to numbers (like 3.14).

Empty strings convert to 0.

Anything else converts to `NaN` (Not a Number).

```
Number("3.14")    // returns 3.14
Number(" ")       // returns 0
Number("")        // returns 0
Number("99 88")   // returns NaN
```

The **parseInt()** will convert a text to an integer

What is an Array?

An array is a special variable, which can hold more than one value at a time.

Creating an Array

Using an array literal is the easiest way to create a JavaScript Array.

Syntax: `var array_name = [item1, item2, ...];`

```
var cars = ["Saab", "Volvo", "BMW"];
```

```
<!DOCTYPE html>
<html> <body>
<p id="demo"> Hi </p>
<script>
var numbers = [];  //declaring an empty array
var i;
var txt = "";
for(i=0;i<=3;i++){
 numbers[i] = prompt("enter a number");
 txt += numbers[i] + "<br>";
}
document.getElementById('demo').innerHTML = txt;
</script> </body> </html>
```

Access the Elements of an Array

You access an array element by referring to the **index number**.

This statement accesses the value of the first element in `cars`:

```
var name = cars[0];
```

```
var cars = ["Saab", "Volvo", "BMW"];
document.getElementById("demo").innerHTML = cars[0];
```

Access the Full Array

With JavaScript, the full array can be accessed by referring to the array name:

```
var cars = ["Saab", "Volvo", "BMW"];
document.getElementById("demo").innerHTML = cars;
```

Array Elements Can Be Objects

JavaScript variables can be objects. Arrays are special kinds of objects. Because of this, you can have variables of different types in the same Array. You can have objects in an Array. You can have functions in an Array. You can have arrays in an Array:

```
myArray[0] = Date.now;
myArray[1] = myFunction;
myArray[2] = myCars;
```

Array Properties and Methods

The real strength of JavaScript arrays are the built-in array properties and methods:

```
var x = cars.length;   // The length property returns the number of
elements
var y = cars.sort();   // The sort() method sorts arrays
```

The length Property

The `length` property of an array returns the length of an array (the number of array elements).

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.length;   // the length of fruits is 4
```

Accessing the First Array Element

```
fruits = ["Banana", "Orange", "Apple", "Mango"];
var first = fruits[0];
```

Accessing the Last Array Element

```
fruits = ["Banana", "Orange", "Apple", "Mango"];
var last = fruits[fruits.length - 1];
```

JavaScript Array Iteration Methods

Array iteration methods operate on every array item.

Array.forEach()

The `forEach()` method calls a function (a callback function) once for each array element. The output will be the contents of the array.

```
<p id="demo"></p>

<script>

var txt = "";

var numbers = [45, 4, 9, 16, 25];

numbers.forEach(myFunction);

document.getElementById("demo").innerHTML = txt;

function myFunction(value) {

 txt = txt + value + "<br>";

}

</script>
```

Adding Array Elements

The easiest way to add a new element to an array is using the `push()` method:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.push("Lemon");    // adds a new element (Lemon) to fruits
```

New element can also be added to an array using the `length` property:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits[fruits.length] = "Lemon";    // adds a new element (Lemon) to
fruits
```

Adding elements with high indexes can create undefined "holes" in an array:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits[6] = "Lemon";    // adds a new element (Lemon) to fruits
```

```
The above array is missing the indexes 4 & 5
```

Removing Array Elements

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.pop(");    // removes the last element of fruits
```

JavaScript Errors - Throw and Try to Catch

- The `try` statement lets you test a block of code for errors.
- The `catch` statement lets you handle the error.
- The `throw` statement lets you create custom errors.
- The `finally` statement lets you execute code, after try and catch, regardless of the result.

Errors Will Happen!

When executing JavaScript code, different errors can occur.

Errors can be coding errors made by the programmer, errors due to wrong input, and other unforeseeable things.

In this example we have written alert as adddlert to deliberately produce an error:

```
<p id="demo"></p>

<script>
try {
  adddlert("Welcome guest!");
}
catch(err) {
  document.getElementById("demo").innerHTML = err.message;
}
</script>
```

Mouse Events

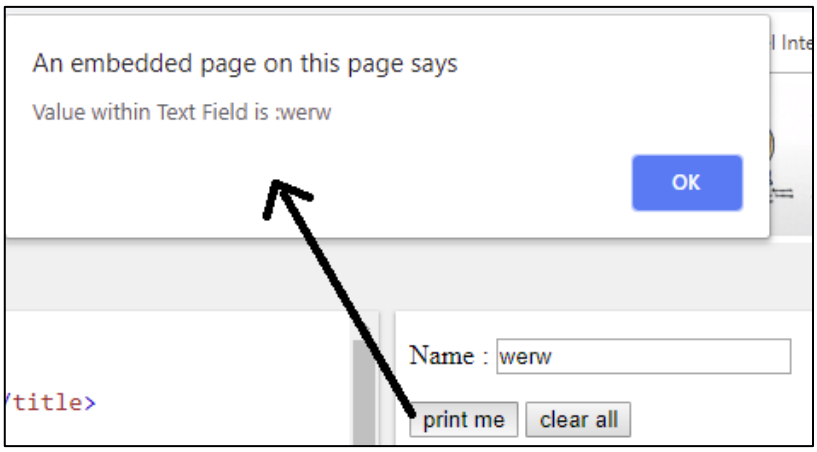
```
<!DOCTYPE html>
<html>
<body>
<h1 onmouseover="style.color='red'" onmouseout="style.color='black'">Mouse over this text</h1>
</body>
</html>
```

On mouse Down

```
<!DOCTYPE html>
<html>
<head>
<script>
function myFunction(elmnt, clr) {
  elmnt.style.color = clr;
}
</script>
</head>
<body>
<p onmousedown="myFunction(this,'red')" onmouseup="myFunction(this,'green')">
Click the text to change the color. A function, with parameters, is triggered when the mouse button is pressed down, and again, with other parameters, when the mouse button is released.
</p>
</body>
</html>
```

Capturing form events

```
<!DOCTYPE html>
<html> <head><title>Javascript Form Events : keyword this</title>
<script>
    function validateF(iform){
      alert("Value within Text Field is :" +
           iform.namestring.value ); }
  </script>   </head>
<body>
  <form name ="demo_form">
   <p>   Name :   <input type="text" name="namestring" size="20" />  </p>
   <input type="button"    value="print me" onclick="validateF(this.form);" />
<!-- Reference to the form object  using this keyword ---->
   <input type="reset" value="clear all">
 </form>
</body></html>
```



Focus and Blur events

The focus event occurs when an element gets focus (when selected by a mouse click or by "tab-navigating" to it).

The focus() method triggers the focus event, or attaches a function to run when a focus event occurs.

The **blur** event occurs when an element loses **focus**. The **blur**() method triggers the **blur** event, or attaches a function to run when a **blur** event occurs.

```
!DOCTYPE html>

<html> <head>

<style>

a:focus, a:active {

  color: green;

}

</style>

</head> <body>

<a id="myAnchor" href="https://www.w3schools.com">Visit W3Schools.com</a>

<p>Click the buttons to give focus and/or remove focus from the link above.</p>

<input type="button" onclick="getfocus()" value="Get focus">

<input type="button" onclick="losefocus()" value="Lose focus">

<script>

function getfocus() { document.getElementById("myAnchor").focus(); }

function losefocus() { document.getElementById("myAnchor").blur();  }

</script>

</body> </html>
```

Validation Check

Validation is a process used to ensure that computer systems will produce information or data that meet a set of defined **requirements**.

- Range check – marks entered is within the accepted range
- Presence check – information has been entered and not left blank

Q-1: create a program to accept a given line of text. The user then enters a particular character that is to be searched in the entered line of text. Display the number of times the given character appears in the line of text. Use loops and the **charAt()** function.

Q.2 Write functions to convert lower to uppercase and upper to lower case. Have a text box to enter the text and two buttons to invoke the functions.

```
<!DOCTYPE html> <html> <body>
<p>Click the button to convert the string to uppercase letters.</p>
<p id="demo"></p>
<script>
function myCapital(myform) {
  var str = myform.txtInput.value
  var res = str.toUpperCase();
  document.getElementById("demo").innerHTML = res;
} </script>
<form name ="demo_form">
<input type="text" name="txtInput" value="Capitalise">
<input type="button" onclick="myCapital(this.form)" value="Capitalise">
</form> </body></html>
```

Q-3: Have two text boxes for the user to enter the username and password. Valid users are Tom and Jack whose passwords are Mary and Jill respectively. The user names and passwords are not case sensitive. Show a message “Welcome” to valid user logins and an error message otherwise.